Evaluating Traffic Redirection Mechanisms for High Availability Servers

Elias Konidis*, Panagiotis Kokkinos*†, and Emmanouel Varvarigos*†

*Dept. of Computer Engineering and Informatics, University of Patras, University Campus, Rion, Greece
†Computer Technology Institute & Press "Diophantus", University Campus, Rion, Greece
*†Email: {konidis, kokkinop, manos}@ceid.upatras.gr

***Abstract*— The availability of cloud applications and their respective server high availability is affected by inefficient load balancing configurations, denial of service attacks (DoS), link failures, power outages etc. As these issues may never disappear, traffic redirection from the affected server to a backup one is an important procedure followed for restoring the provided service. In this work, we initially evaluate a reverse proxy server-based scheme for traffic redirection, using High Availability proxy server(HAproxy) and also we develop a Software Defined Network (SDN)-based solution. We compare the proxy server-based and the SDN-based schemes, exhibiting how each can be used to mitigate downtime in case of a failure, providing high availability.**

***Index Terms*—Reverse Proxy Server , Software Defined Network, High Availability, OpenFlow.**

I. Introduction

The failure of servers that support cloud applications, usually translates to unsatisfied customers, revenue losses and damaged reputation. The International Working Group on Cloud Computing Resiliency (IWGCR) [1] has recorded outages in 38 well-known cloud services, that have cost the economy about $700 million dollars.

Several mechanisms have been developed to provide high availability against such failures, which are usually based on the maintenance of an active/primary and a passive/backup server. Data between the primary and the backup servers are made consistent either synchronously or asynchronously. When a failure occurs, the backup server becomes the primary and the traffic originally destined for the primary server must be redirected to the backup, by configuring appropriately the network infrastructure. Traffic redirection is thus an important part of the whole high-availability equation, since it directly affects the transparency of the failure recovery operations.

Traffic redirection techniques have been present for many years now, employed in almost all the layers of the ISO/OSI model, mainly in order to serve users that move from one location to another, web services provided from different data centers and failures. Traffic may need to pass via different types of networks (WAN, LAN) when a redirection occurs. So, the relative position of the primary server, the backup server and the client plays an important role. Traffic redirection mechanisms target both node and link failures. In the node failure scenario the main issue is to determine the backup server that will handle the load of the failed server. In the link failure scenario the important issue is the recomputation of the path over which traffic will be forwarded to the server. Reducing the convergence time in computing the restoration path is critical. Also, if a link fails, and depending on the infrastructure, the topology may change dramatically, so one has to check if another (backup) server is closer to the client. Another important consideration is the nature of the protocol that is redirected. Different mechanisms must be deployed for stateless protocols (like UDP) and different ones for protocols that need to keep the state of a session (like TCP/HTTP). The main difference is that stateless protocols do not need to establish a connection (3 handshake) among participants.

Our main goal is to experimentally depict the differences among the way a classic reverse proxy server and a SDN-based architecture can handle server failures. Towards this end, we implemented both architectures in a common emulation environment. In that way our results depicts purely the differences among them. Metrics that are used here are the servers' downtime, the number of clients aborted due to unavailable service, and the average throughput achieved by the clients. The rest of this paper is structured as follows. In Section II we report on previous work. In Section III, we describe the proxy server scheme and present the SDN-based traffic redirection mechanism we developed. In Section IV we describe the mechanisms implementation details and present the results of the performed evaluation. Finally, in Section V we conclude the paper.

II. Related work

Previous work includes traffic redirection handling at different OSI layers, mostly the transport layer and the application layer. Some of this work is widely used in practice while other work corresponds to standard tracks or experimental extensions on existing standards.

A. Legacy protocols

The network layer BGP-based traffic redirection mechanism aims to supply routing tables with abundant routing information. For that reason the anycast routing scheme is used, with every router advertising its networks, so that any route's Routing Information Base(RIB) keeps an alternative path for them. This approach is promising as it can also provide load balancing, but suffers from the route flap problem [2] that needs to be taken into account. Content Distribution Network

978-1-5090-2482-7/16/$31.00 ©2016 IEEE

(CDN) CloudFlare [3] [4] in the WAN level provides a solution based on BGP Anycast.

At the transport layer there are two protocols that posses features, that can be used for traffic redirection: the Stream Control Transport Protocol (SCTP) [5] and the Mulipath-TCP (MPTCP) [6]. Both are related with TCP, ensuring surefooted and in sequence delivery, providing a reliable transport service. SCTP is message-oriented instead of TCP byte-oriented. That enables two useful features of SCTP, Multi-streaming and Multi-homing. The first partitions the data into more than one streams (in contrast to TCP, which uses one stream), providing fault tolerance, as the failure of one stream does not affect the other streams and the transportation mechanism. Although SCTP is a unicast protocol, the endpoints participating the session may be represented by multiple IP addresses (one chosen for the primary), providing the Multi-homing feature. SCTP handles path and endpoint failures by maintaining counters. Retransmissions are sent to another endpoint of the list that have been exchanged and if a configured maximum is exceeded, all data are redirected to that. An experimental protocol useful for redirection is MPTCP. Connection initiator can use all of its available interfaces to create multiple paths to an endpoint server. In that way sub-flows are created that provide resilience to network failures and higher throughput.

B. Software Defined Networking

Among the more novel approaches on the high availability and traffic redirection problem are those using the Software Defined Networking (SDN) paradigm. Software Defined Edge (SDE) [7] is an implementation of SDN at network Edge. An interesting part on this work is the mechanism for inline TCP duplication. The main idea is to establish paths to both servers (primary and backup) and compute the difference between the Initial Sequence Numbers assigned to them according to the TCP handshake. Then, the SDN Controller can change the TCP headers of the backup connection by using that offset. In this way, when the primary server fails, the client receives a response from the backup. The problem with this process may be the huge traffic that the controller must handle to keep backup synchronized, when all the connections need to be treated in that way. Also, as the authors in [7] themselves mention, SDE deployment raises new challenges on the partnership between ISPs and Content Distribution Network (CDN) providers.

III. Traffic Redirection Schemes

In this section we compare existing methods for traffic redirection with novel ones. We describe the main functionality of HAProxy and propose a general algorithm for fast TCP traffic redirection, using an SDN scheme.

A. Reverse Proxy Server based

HAProxy, which stands for High Availability Proxy [8], is primarily used for load balancing in the cloud. A lot of known organizations use this open source solution. Among others one of HAProxy advantages is that it is flexible and allows the redirection of traffic based on events and internal status. To achieve this, a health checking mechanism is needed, and can be used both on Layer 4 and Layer 7. We focus on Layer 4 traffic redirection.

A reverse proxy server acts on behalf of service server. Each session splits in two connections, on connection between a client and the proxy and the other between the proxy and the application server. In that way the proxy can handle connections, using inbound and outbound rules and providing high availability. In particular, HAproxy provides highly configurable health-checking mechanisms (e.g for MySQL, SMTP). Health checking mechanism uses TCP, where a 3 handshake establishment testifies that the server is up. That health-checking mechanism is using the same physical mean the data path flows. So HAproxy needs to accomplish a number of tries (3 tries by default) in order rightly ruling for a failure. Also, the concept of splitting session on two connections leads to reduced throughput. Once the failure is detected and network responds, backup server's info are used for future connection establishment, as Figure 1(a) depicts. The simplicity of integration and the fact that the network handles traffic destined for the backup exactly the same way with that destined for the primary server, constitute advantages of this approach.

B. Software Defined Networking

Software-Defined Networking (SDN) [9] promises the separation of the network control plane from the forwarding plane, and the ability of the control plane to manage several network devices. This decoupling of network control from the forwarding functions enables the network to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. OpenFlow [10] is the de facto communication protocol between the control and forwarding layers of an SDN-based networking, which carries the forwarding related decisions made in a centralized OpenFlow controller. In this way, OpenFlow allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based).

We extend the OpenFlow Controllers logic, implementing a Simple_TCP_Redirection module, so as to handle server failures and initiate traffic redirection between the primary and the backup server. The main functionality of the module is described by Algorithm 1 and is based on the topology presented in Figure 1(b).

An OpenFlow controller uses a number of standard messages to install/delete/modify forwarding flows in the OpenFlow enabled forwarding devices. For our purposes, among the 34 different OpenFlow message types(OFP 1.4), we have to focus on three of them: **PacketIn** (type code 10), **PortStatus** (type code 12) and **FlowMod** (type code 14). The PacketIn message is a way for the OpenFlow enabled switch to send a captured packet to the controller. This can happen either because of a miss in the matching tables or because it is an explicit action as a result of a set action asking for this

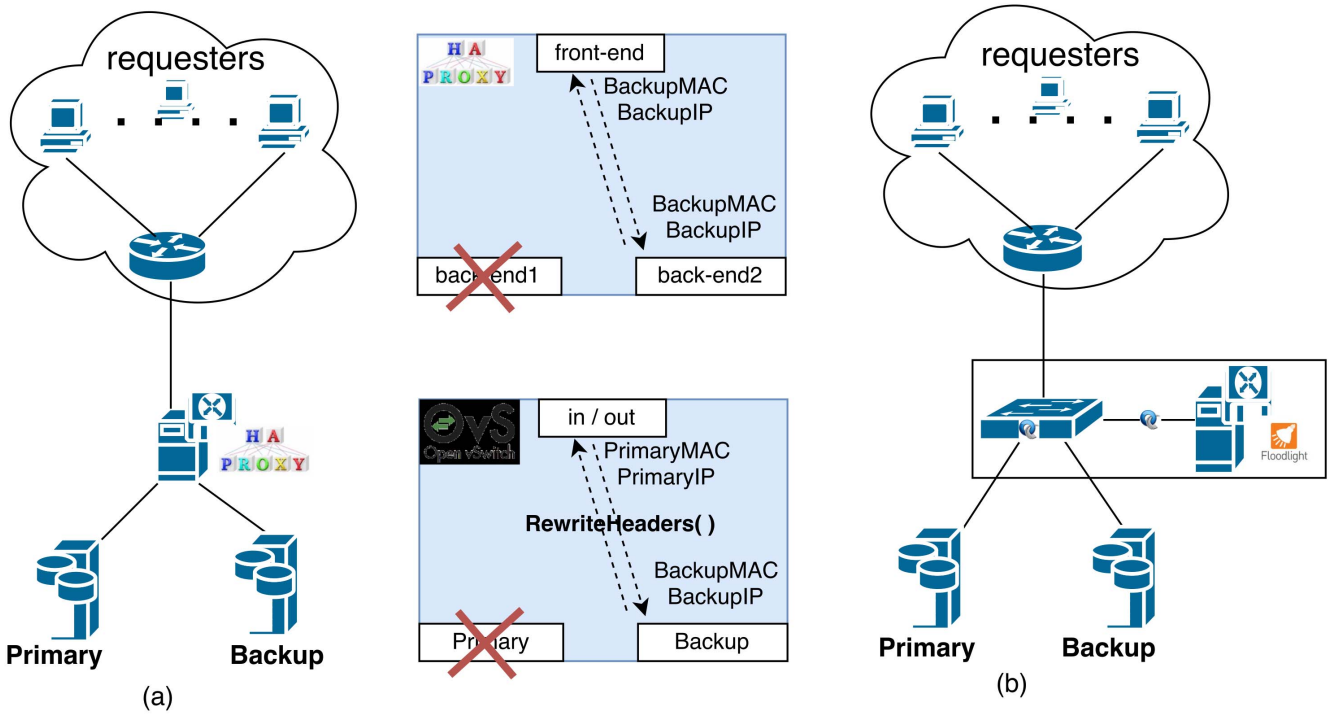

Fig. 1: (a) HAProxy-based redirection (b) SDN-based redirection

behaviour. PortStatus messages are asynchronous events sent from the switch to the controller indicating a change of status for the indicated port. FlowMod is one of the main messages, which allows the controller to modify switch Flow table.

Another basic concept is that of the exact matching. We can create matching based on different protocols headers. If a packet enters dataplane and the matching criteria are fulfilled, the packet is forwarded to the next dataplane device. Otherwise, that packet is encapsulated in a PacketIn OpenFlow message and sent to the controller. The important thing to consider here is that we can create different packet handling classes, based not only on fulfilled matching criteria, but also on one that not are fulfilled. We can categorize packets in two general classes: Already forwarded(due to match fulfillment) and PacketIn(due to mismatches).It is critical to use this approach for efficient control plane applications development.

Algorithm 1 presents the basic functionality of traffic redirection. The basic idea is to use a mechanism able to provide a transparent procedure from the client's side. To this end, we used two procedures, the **SwitchPortChanged()** and the **Receive()**. The Receive() method is over-riden in order to handle new **PacktIn** messages. In the normal case TCP flows based on IP,MAC,port shall be installed on **dataplane**, to provide connectivity among client and server. So packets entering **dataplane** are forwarded with that (IP,MAC) pair on primary server. Also packets destined to client have as source (IP,MAC) pair the one of the primary. Upon a failure occurs the **SwitchPortChanged()** method deletes all the flows that are leading to clients/primary server. This flow removal is critical, as the faster the flows leading to the black holed primary interface are removed, the faster upcoming connections will not be forwarded. Such a mismatch will lead to **PacktIn** messages. Then, the controller, takes care for a special action in order to install flows responsible to rewrite server's IP and MAC. Specifically, as Figure 1(b) depicts, packet headers entering **dataplane** are using primary server info. Then flows, rewrite the destination (IP, MAC) pair and forward them to the port attached to the backup server. The reverse action takes place when backup server responds, rewriting source fields at packet headers and providing transparency.

The preceding method provides fast/transparent redirection, without splitting a session into two connections. That type of handling also provides high throughput, as the results section will indicate. But a basic disadvantage of this approach is that once a failure occurs, a lot of network resources are consumed due to header rewrites. Because of the huge overhead for packet header rewrites after a failure occurs, this algorithm fits best to environments where the primary server becomes available again in a short time.

IV. Implementation And Evaluation

A. Implementation

To implement the proposed mechanisms we used the Mininet network emulator [11], which is based on network namespaces, depicting purely the differences among the proposed mechanisms in common base. We use a python script to create the topologies of Figure 1, utilizing Mininets python API. This python script allows us to easily expand the

Algorithm 1 Simple_TCP_Redirection

```
1: p1 ← port attachedToPrimeServer
2: p2 ← port attachedToBachup
3: p3 ← port Gateway
4: procedure SwitchPortChanged( swId, port, type)
5:     if PORT == p1 && TYPE == down then
6:         //packets destined to failed server
7:         match_in ← ExactMatch(
               Dstport(service_port),
               DstMAC(PrimMac));
8:         //packets coming from failed server
9:         match_out ← ExactMatch(
               Srcport(service_port),
               SrcMAC(PrimMac));
10:        FLOW_MOD(match_in, match_out, DELETE)

11: procedure Receive(sw, msg)
12: if MSG.type.equals(PACKETIN) then
13:     TCP ← MSG.getIP().getTCP()
14:     //Normal traffic handling
15:     if SW.portEnabled(p1) then
16:         if TCP&&TCP.dstport(service_port) then
17:             match1 ← ExactMatch(
                    SrcClient(port, IP, MAC),
                    DstServer(port, PrimIP, PrimMAC);
18:             match2 ← ExactMatch(
                    SrcServer(port, PrimIP, PrimMAC),
                    DstClient(port, IP, MAC);
19:             //Install flows
20:             FLOW_MOD(match_in, OUT(p1))
21:             FLOW_MOD(match_out, OUT(p2))
22:         else
23:             //other service handling
24:     else
25:         //Redirected traffic handling
26:         if TCP&&TCP.dstport(service_port) then
27:             match_in ← ExactMatch(
                    DstIP(PrimIP),
                    DstMAC(PrimMAC));
28:             Action_in ← RewriteHeaders(
                    DstIP←BckupIP,
                    DstMAC←BckupMAC);
29:             match_out ← ExactMatch(
                    SrcIP(BckupIP),
                    SrcMAC(BckupMAC));
30:             Action_out ← RewriteHeaders(
                    SrcIP←PrimIP,
                    SrcMAC←PrimMAC,);
31:             //Install flows
32:             FLOW_MOD(match_in, Action_in, OUT(p2))
33:             FLOW_MOD(match_out, Action_out, OUT(p3))
34:         else
35:             //other service handling
```

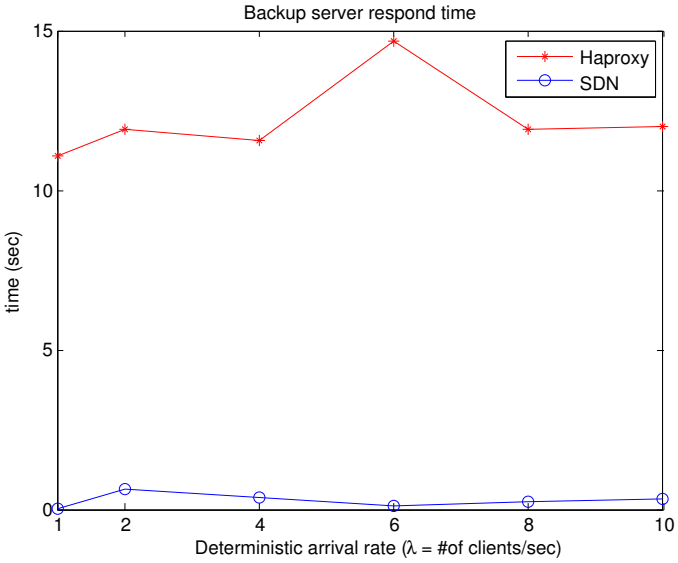

Fig. 2: Backup's respond time.

topology for further experiments. The main functionality of the device connected to clients is to provide connectivity to servers(acting as Gateway) enabling a dynamically growing number of requests. The unicast routing system is used for all devices. Link characteristics(delay, packet loss, capacity, queues) among access points(HAproxy/openvswitch) are left as configured by Mininet by default. In both experiments we used iperf [12] as TCP traffic generation tool forcing each connection transmit at most 10 sec. To emulate the failure we put down the primary server's interface, 5 sec after the experiment began. In order to measure the backup's response time, we used tshark [13], to capture appropriate packets on respective interfaces.

In order to use HAproxy we configured a Mininet node on its namespace. A node provides three interfaces: A front end, for clients access, and two back-ends for the primary and backup server. We use linux commands providing Layer 3 connectivity among clients and servers. Moreover, for HAproxy, we use a simple configuration file with heart-beat check set to 3 sec.

The experimental topology for the SDN-based traffic redirection consists of a OpenVswitch(OVS) [14], two servers (primary and backup) and a cluster of requesting hosts (Figure 1). For the control plane logic, we used the Floodlight controller [15] (using loopback interface) and the implemented Simple_TCP_Redirection module (Algorithm 1) that overrides some of its methods.

B. Results

In order to compare these redirection schemes we used metrics that reflect the transparency that clients need. Backups response time is one of the most important metrics, as it reflects how fast the network can respond to failures. As already mentioned, both mechanisms have an almost constant value for the response time when client's arrival rate increases.

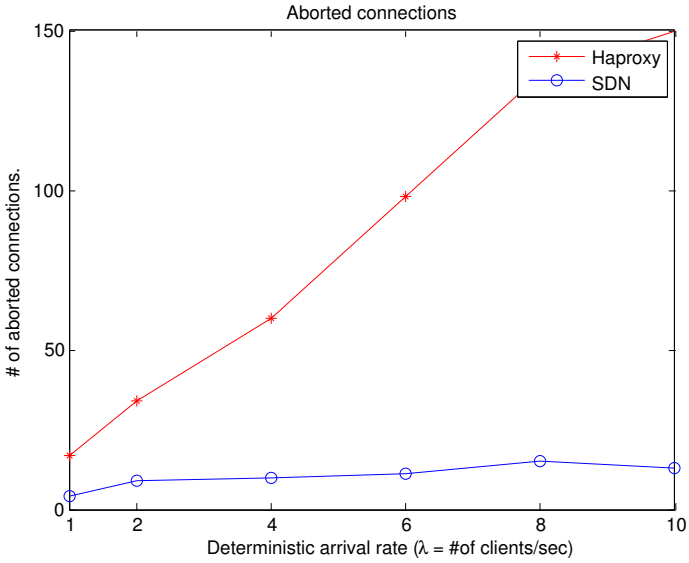

Fig. 3: Number of TCP RST messages due to failover.

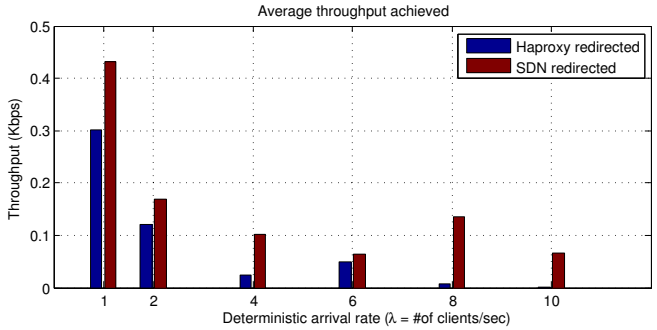

Fig. 4: Average throughput achieved.

The SDN based scheme, by using Openflow messages, responds in some hundreds of milliseconds, while HAproxy, by using a couple of 3 hand shake messages for health-checking, increases significantly the response time, as Figure 2 depicts.

Downtime affects immediately how many clients will find the service up. Especially, for the requests (SYN messages) sent between the primary's failure and the backup's response time, the downtime is critical as TCP in most implementations tries 5 times (using SYN Retransmissions) if SYN timers expire. That means that a fast redirection leads with higher probability to a 3 handshake establishment. Figure 3 illustrates that as the arrival rate increases, HAproxy aborts connections in a linear way, while SDN keeps the number of aborted connections relatively constant.

Our final metric is the average throughput achieved by the clients as the arrival rate increases. Under normal conditions (not a failure scenario), SDN architecture achieves significantly higher throughput. A basic reason is that reverse proxies have to split the initial connection in order to achieve corresponding functionality. On the other hand, SDN handles PactIn messages on the fly, creating appropriate flows and corresponding functionality. Figure 4 depicts SDNs high throughput even in a failure scenario. HAproxy experiences large downtime and more retransmissions, significantly affecting the average throughput experienced by the clients.

V. Conclusion

Traffic redirection mechanisms are important in achieving high availability. When a (link/node) failure occurs, proxy based schemes use health-check mechanisms and routing protocols exchanging messages based on timer expirations. Software Defined Network architecture proved to be very robust, thanks to the ability of the SDN controller to supervise network interfaces. Also, openflow-enabled switches are able to send asynchronous messages about the state of their ports and adjacent links.

In addition to high availability, traffic redirection mechanisms are useful for other networking purposes, including load balancing. When a server of a cluster is overloaded, a mechanism to pass traffic from one server to another is useful and even critical, when servers are stressed. Monitoring when a server reaches its limits is of course necessary, but achieving zero traffic redirection can prevent crashes and miss behaviors. Another case is Denial of Service attacks (DoS), where traffic redirection can be used either for healing affected connections or for reducing the size of the attack.

We should note that a big challenge is not just to reduce the downtime of service and the number of aborted connections, but to also develop a well-defined mechanism for live-redirecting affected connection and keeping the TCP state.

Acknowledgement

This work has been supported by the European Commission through the FP7-ORBIT project under grant agreement number 609828 and the H2020-NEPHELE project under grant agreement number 645212.

References

[1] Christophe Cerin, Camille Coti: *Downtime Statistics of Current Cloud Solutions, 2014*
[2] Route flapping: $https://en.wikipedia.org/wiki/Route\_flapping$
[3] Anycast, CloudFlare blog: *http://blog.cloudflare.com/a-brief-anycast-primer*
[4] Load Balancing without Load Balancers, CloudFlare blog: *http://blog.cloudflare.com/cloudflares-architecture-eliminatingsingle-p/*
[5] RFC 4960: *Stream Control Transmission Protocol, 2007*
[6] RFC 6897: *Multipath TCP (MPTCP) Application Interface Considerations, 2013*
[7] V. Gramoli, G. Jourjon, O. Mehani: *Can SDN Mitigate Disasters?*
[8] The Reliable, High Performance TCP/HTTP Load Balancer: *http://www.haproxy.org/*
[9] Open Networking Foundation(ONF): *SDN architecture, 2014.*
[10] Nick McKeown, Scott Shenker, Hari Balakrishnan: OpenFlow: EnablingInnovation *in Campus Networks, ACM SIGCOMM Computer Commu- nication Review, Vol. 38, No. 2, 2008.*
[11] Bob Lantz, Brandom Heller, Nick McKeown, A Network in a Laptop: *Rapid Prototyping for Software-Defined Networks, ACM SIGCOMM Workshop on Hot Topics in Networks, 2010.*
[12] iperf: *https://iperf.fr/*
[13] tshark: *https://www.wireshark.org/docs/man-pages/tshark.html*
[14] Openflow virtual Switch(OvS): *http://www.openvswitch.org/*
[15] Project Floodlight: *http://www.projectfloodlight.org/floodlight/*