International Journal of Trend in Research and Development, Volume 3(2), ISSN: 2394-9333
www.ijtrd.com

High Availability and Load Balancing in SDN Controllers

Ankit Rao, Shrikant Auti, Akhil Koul and Gauri Sabnis,
Department of Computer Engineering, Pune Institute of Computer Technology,
Savitribai Phule Pune University (Formerly University of Pune), Pune, Maharashtra, India

***Abstract*—**SDN is an emerging computer networking paradigm that supports programmable interfaces which provide an agile and convenient way to customize the network traffic control. The main contribution of the SDN architecture is the disaggregation of the vertically integrated networking planes viz. the control plane and the data plane in order to improve network flexibility and manageability. The control plane makes decision about where the traffic is sent. The control plane function includes the system configuration, management and exchange of routing table information. The data plane also known as the forwarding plane forwards traffic to the next hop along the path to the selected destination network according to the control plne logic. In SDN network based on OpenFlow a controller performs logically centralized control of enterprise network infrastructure, network policies, and data flows. At the same time the controller is a single point of failure which can cause a very serious problem (e.g. network outage) for network reliability and production use cases. Also, the controller failure may occur due to the improper distribution of data traffic. The controllers, though are able to handle good amounts of traffic but they crash if the traffic changes are unexpected. To address this problem, we consider different active/standby strategies to provide a controller failover in case of controller failure. We propose a high-available controller (HAC) architecture, which allows us to deploy a high availability control plane for enterprise networks. Also, we implement a load balancing strategy to distribute the traffic among the servers.

Keywords*—Router, Software Defined Networking, Open Flow, Openvswitch, Control Plane, SDN Controllers, Load Balancing, High Availability, Network Management.*

I. INTRODUCTION

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. Here, the control plane is consolidated into a centralized controller that uses the OpenFlow protocol, or alternate communication methods, to control each node and traffic flow on the network. SDN is directly programmable, agile, centrally managed, programmatically configured, open standards based and vendor neutral.

The implementation involves deployment of the control plane of the switch onto one or more SDN controllers running on dedicated servers thus giving the controllers a complete view of the topology of the network. The controller then customizes the network traffic control. In spite of the SDN advantages, one of the serious problems of SDN is that the controller is a critical point of failure and, therefore, the controller decreases overall network availability .A controller failure can be caused by various reasons: failure of the server where a controller is running, the server operating system failure, power outage, abnormal termination of the controller process, network application failure, network attacks on the controller and many others.

An approach for improving the SDN control plane availability in case of a controller failure in the enterprise software-defined networks is presented in the paper. We implement cluster paradigm accounting multiple controllers in the controller plane for solving the controller failure problem. Multiple controllers are situated but for the switches in the lower plane these multiple controllers are logically equivalent to one controller. This is achieved by using a virtual IP address and MAC address. In case of a primary controller failure the election algorithm elects the new primary controller providing high availability.Also, a load balancer for equally distributing the traffic load on the servers providing common service is combined with the high availability of the architecture which aims at increasing the uptime of the system to a greater extent.

II. LITERATURE SURVEY

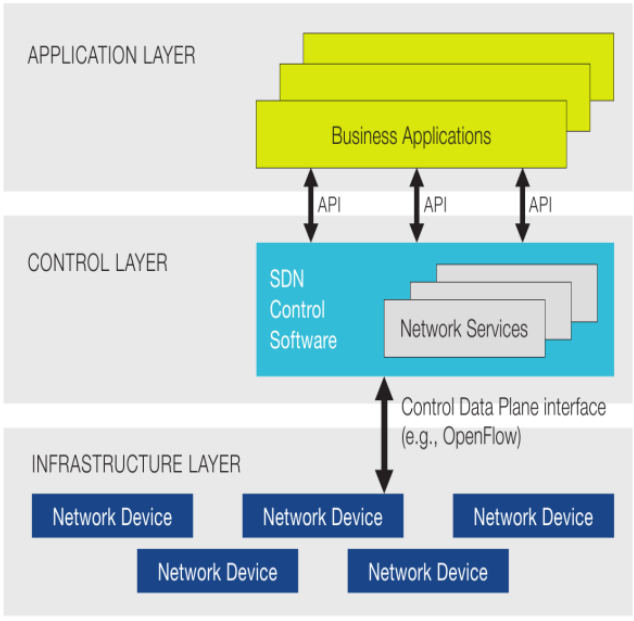

Figure 1. SDN Architecture

SDN Architecture: Traditionally, both the control and data plane elements of a networking architecture were packaged in proprietary, integrated code distributed by one or a combination of proprietary vendors. The OpenFlow standard created in 2008, was recognized as the first SDN architecture that defined how the control and data plane elements would be separated and communicate with each other using the OpenFlow protocol. In the SDN architecture, the splitting

of the control and data forwarding functions is referred to as "disaggregation," because these pieces can be sourced separately, rather than deployed as one integrated system.

This architecture gives the applications more information about the state of the entire network from the controller, as opposed to traditional networks where the network is application-aware.

SDN architectures generally have three components or groups of functionality:

1. SDN Applications: SDN Applications are programs that communicate behaviors and needed resources with the SDN Controller via application programming interface (APIs). In addition, the applications can build an abstracted view of the network by collecting information from the controller for decision-making purposes. These applications could include networking management, analytics, or business applications used to run large data centers. For example, an analytics application might be built to recognize suspicious network activity for security purposes.
2. SDN Controller: The SDN Controller is a logical entity that receives instructions or requirements from the SDN Application layer and relays them to the networking components. The controller also extracts information about the network from the hardware devices and communicates back to the SDN Applications with an abstract view of the network, including statistics and events about what is happening.
3. SDN Networking Devices: The SDN networking devices control the forwarding and data processing capabilities for the network. This includes forwarding and processing of the data path. The SDN architecture APIs are often referred to as northbound and southbound interfaces, defining the communication between the applications, controllers, and networking systems.

Northbound API: It sits to the top of the controller. The northbound API presents a network abstraction interface to the applications and management systems at the top of the SDN stack. The information from these applications is passed along through a southbound interface.

Southbound API: southbound application program interfaces (APIs) are used to communicate between the SDN Controller and the switches and routers of the network. They can be open or proprietary .Southbound APIs facilitate efficient control over the network and enable the SDN Controller to dynamically make changes according to real-time demands and needs. OpenFlow, which was developed by the Open Networking Foundation (ONF), is the first and probably most well-known southbound interface. It is an industry standard that defines the way the SDN Controller should interact with the forwarding plane to make adjustments to the network, so it can better adapt to changing business requirements. With OpenFlow, entries can be added and removed to the internal flow-table of switches and potentially routers to make the network more responsive to real-time traffic demands.

III. GENERAL ARCHITECTURE

Current researches on software defined networking (SDN) still face many challenges such as the performance of control plane, high availability of controller, scalability, consistency issues of network states, especially high availability of network controller. OpenFlow protocol supports different kinds of controller roles. Switch can connect to both a master controller and multiple slave controllers. If the primary controller fails, a new primary controller would be elected from the cluster of controllers. High-available controller (HAC) architecture is based on adding of additional cluster middleware between the controller core and controller network services and applications. To provide fault-tolerance of the control platform, the HAC cluster middleware includes the following managers and services:

Managers:

1. Controller Manager to coordinate start/restart/stop controller network services and applications and up and down control interface for network devices connections.
2. Cluster Manager to control the operation of the controllers cluster and distribute responsibilities (primary or standby) in accordance with the cluster configuration file.
3. Sync Manager to control controller network services and applications synchronization between controller instances in the cluster.
4. Recovery Manager to coordinate the recovery process (failover and fail- back) in case of controller instance failure in the platform.

Services:

1. Message Service to provide control message distribution to other controller instances in the controller cluster.
2. Event Service to provide filtering, distribution and processing to or from other controller instances.
3. Heartbeat Service to monitor the operational status of the controllers and detects controller failures in the controller cluster.

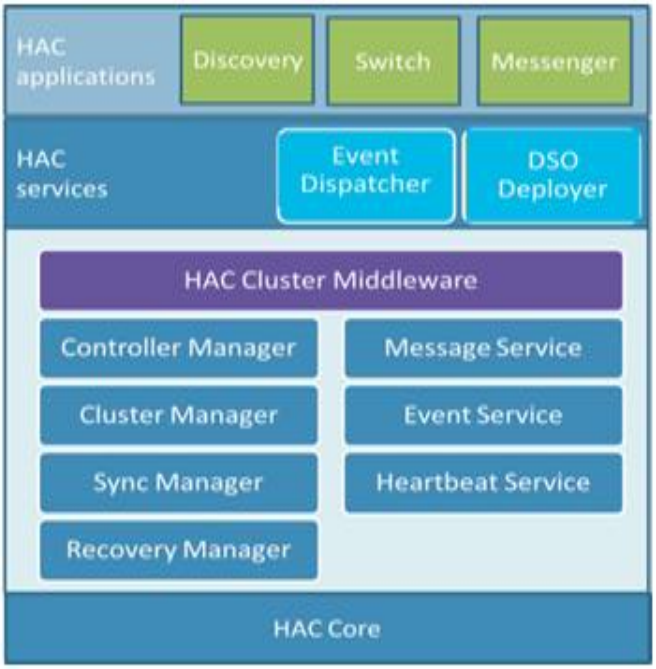

Figure 2: High availability architecture

IV. PROPOSED METHOD

A. High Availability

In order to eliminate single point of failure in Software Defined Networks and ensure reliability we propose an approach using the cluster paradigm since the cluster paradigm has number of advantages over the master slave paradigm.

1. No overhead of manual configuration for all controllers on each device.
2. Increased High availability as there are multiple controllers available if primary fails.
3. Only single IP address is enough to deploy multiple controllers in cluster.

The approach involves setup of a cluster of controllers above the switches in a network. The entire cluster of controllers is represented by a single virtual IP that would be the IP of the primary controller. The primary controller will be elected based on the priorities assigned to the individual controllers. The primary controller will be responsible for network data flows and the other controllers will keep in synchronization with the primary controller. As the primary controller failure occurs the election algorithm is executed to determine the next primary controller.

1. Initial Mode: The system starts with the data flow through the network. The primary controller (according to priority) takes control of the network. Other controllers in the cluster establish a connection to the primary controller via the proposed Northbound API .Other controllers request the current Network view and network interfaces list for control channels connections, current states of network services and applications to the primary controller.

2. Operational Mode: In this mode primary controller processes OpenFlow messages from network devices and controls network data flows, the standby controllers monitor the primary controller state and synchronize with it. The controller state includes the network topology view, network services and applications and controller data synchronisation.

3. Failure Mode: This mode consists of two stages:

1. Failure detection- Heartbeat messages are used to detect failure of the primary controller.
2. Recovery- This is initialised after failure is detected. This includes
    a. Election of new primary controller using the election algorithm.
    b. The new primary controller informs about this change to the other controllers and the virtual IP change.
    c. Controller network services and application restoration.
    d. Control network interfaces up.

The election algorithm is executed simultaneously on all the controller nodes of the cluster after failure detection. The output of the algorithm is that the controller with the highest priority is elected as the new primary controller. The algorithm designed is as follows:

```
Algorithm Select_Controller(n)

//Problem Description: This algorithm selects the
appropriate controller based on priority.
//Input: Number of controllers
//Output: Appropriate selected controller

if (this.role == NULL) then
    while (this.role == NULL) do
    multicast this.priority
    check CIBroadcaster.priority;
if (!CIBroadcaster || this.priority>CIBroadcaster.priority)
then
    this.role ← CIBroadcaster
    this.IP ← virtual IP
else
    this.role ← backup
    else if (this.role == CIBroadcaster) then
while (this.role == CIBroadcaster) do
    multicast heartbeat packets
    Listen to new.priority
 if (received new.priority) then
    send this.priority
 if (this.priority < new.priority) then
    this.role ← backup
    else if (this.role == backup) then
    listen to heartbeat packets
 if (no heartbeat packets) then
    this.role ← NULL
```

Where,
this.role = the role assigned to the controller
this.priority = the priority of the controller
this.ip= IP address of the controller
CIbroadcaster = primary controller
CIBroadcaster.priority = priority broadcasted i.e. the priority of the failed controller
new.priority = priority of newly elected primary controller.
This election algorithm deals with the controller failure and thus high availability will be achieved.

B. Load Balancer

The load balancer architecture will be implemented along with the SDN-HA. The main objectives of the load balancer will be as follows:

1) Distribute the incoming data from the network devices into batches. IP batching will be used.

2) The setup consists of multiple servers which provide the same service. Then there are hosts all of which request for the same service and also there is a switch and the controllers with HA architecture.

3) The controller here is programmed such that it enforces the switch to direct packets to the multiple servers in a round robin fashion so that the load is equally distributed among the servers and no server goes down due to excessive-load.

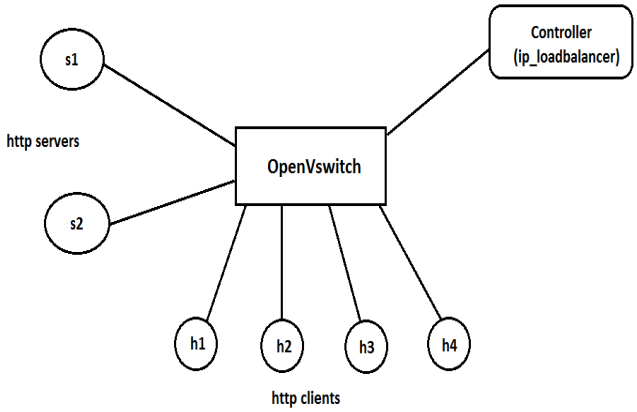

Figure 3: Load Balancer Architecture

Figure 3 illustrates the functioning of the load balancer wherein the requests from the hosts h1, h2, h3, h4 are equally distributed to the servers by the controller.

The working of the load balancer is as follows:

Preemptively ask for the MAC addresses of all the servers with crafted ARP requests, in order to associate these MAC addresses and the corresponding switch ports with the real IP

addresses of the servers. This query is performed upon connection establishment with the switch in order to avoid hanging client flows waiting to be forwarded to the correct server. The ARP replies by the servers will be handled as part of the packet-In handler. Answer to ARP requests from the clients searching the MAC of the service IP, with proxied ARP replies that answer with a fake MAC that is associated with the load balancer (we use "0A:00:00:00:00:01" for simplicity).

It is useful to store the information contained in the ARP request (source MAC address of client, input port of ARP request packet). In this way, when the load balancer needs later to direct flows towards the clients, it will know their MACs and ports to output the packets.

Answer to ARP requests from the servers searching the MAC of a client IP, with proxied ARP replies that answer with the fake MAC that is associated with the load balancer. At this point we already know the MAC of the client, since it has previously requested the MAC address of the load balancer direct flows from the clients towards the servers using the following load balancing mechanism: for each new IP flow from a client, select a server at random and direct the flow to this server. Of course, the server should see packets with their MAC address changed to the MAC of the load balancer, but with the source client IP intact. The destination IP address is rewritten to the one of the server. Be careful: the redirection should only happen for flows that stem from client IPs (i.e., non-server IPs) and which are directed to the service IP. Direct flows

from the servers to the clients, after rewriting the source IP address to the one of the service IP and the source MAC address to the load balancer fake MAC. In this way, the clients do not see any redirection happening, and they believe that all their communication takes place between their MAChines and the service IP.

C. Overall System

The proposed system consists of a high availability architecture consisting of a cluster of controllers with a single virtual IP which will also be responsible for equal distribution of the traffic load to the appropriate servers so that the network uptime increases substantially.

The main reasons for this are increase in the uptime of the servers along with the highly available controllers. Figure4 illustrates the proposed architecture with High Availability(HA) and Load Balancer(LB) features.

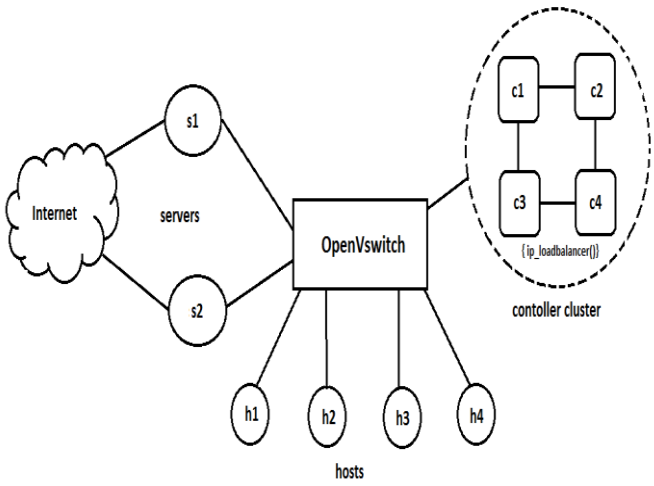

Figure 4: HA and LB architecture

D. Basic Flow

The main aim of the system is to rectify the flaws in the current SDN controllers and networks controlled by them. With that perspective, the high availability architecture with the combination of controller as a load balancer has been designed which substantially tends to improve the uptime of the system.

The data of the system is generated in the form of request/response of services required by the hosts and provided by the servers. The request first goes to the network devices present in the network. Here, we consider the switch as the network device which checks the path for the demanded request in its routing table first. If present, it routes the request appropriately otherwise it queries the controller for the path. The controller provides the path to the switch as it has the overall topology view. In case of the failure of the master controller, the election algorithm is used to elect the new controller and the process continues in the normal fashion again. Also, while returning the response to the requests generated care is taken that the load is equally distributed among the servers providing the same service so that the server does not go down due to excessive load. Thus a highly available network which tends to be reliable is the guaranteed output of the designed method.

The overall system working can be illustrated by the following data flow diagram (level 1)Figure 5.

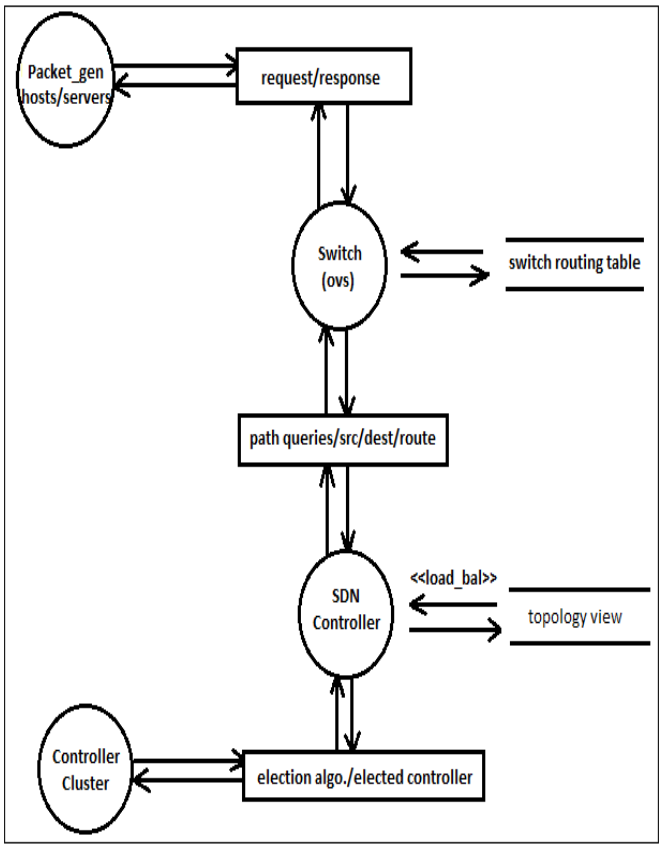

Figure 5: Data flow diagram(level 1)

The following functionalities are expected from the system:

1) Configuration of a few controllers in SDN networks rather than configuring a large number of switches in the traditional networks which is a cumbersome task.

2) Very less down time due to the high availability architecture. Also, the down time caused is due to the time for the election of the new master controller.

3) Load distribution among servers providing similar services so that the servers do not crash due to excess load of packets.

CONCLUSION AND FUTURE SCOPE

The Software-Defined Network (SDN) has recently emerged to address the problem of the ossified Internet protocol architecture and to enable agile and flexible network evolvement. SDN, however, heavily relies on control messages between a controller and the forwarding devices for the network operation. Thus, it becomes even more critical to guarantee network high availability (HA) between a controller and its forwarding devices in the SDN architecture since the controller being the single point of failure.

With extensive experiments using real systems, we have identified that the significant issues of HA in operations of a SDN such as single point of failure of multiple logical connections, multiple redundant configuration, unrecoverable interconnection failure, interface flapping, new flow attack, and event storm. We will be designing and implementing the management frameworks that deal with SDN high availability and scalability issues by using the proposed approach thus eliminating the single point of failure in SDN architecture. Also the load balancer in addition to the high availability architecture helps in decreasing the down time of the network.

Future research would aim at developing better algorithms and strategies to achieve the current developments. The method designed now has some drawbacks resulting due to the redundancy of data packets. Load balancer with respect to CPU utilization is being worked on. Also, developing a load balancer for the controllers would be a great addition to the current development.

Acknowledgment

We would like to thank our guides Prof.P.A.Jain of Pune Institute of Computer Technology and Mr.Vijay Jadhav from the industry working in GS Labs, Pune for their immense support and guidance. Their suggestions and their experience in this field helped us a lot.

References

[1] ONF, Software-Defined Networking: The New Norm for Networks, white paper, https://www.opennetworking.org

[2] Feng Wang, Heyu Wang, 2014, A Research on Carrier-grade SDN Controllers.

[3] V Pashkov, A Shalimov, R Smeliansky, Lomonosov Moscow State University, Controller failover for SDN enterprise networks.

[4] Yi-Chen Chan, Kuochen Wang , Yi-Huai Hsu, Fast Controller Failover for Multi-domain Software-Defined Networks.

[5] Junjie Zhang, Kang Xi, Min Luo, H. Jonathan Chao , 2014, Load Balancing for Multiple Traffic Matrices Using SDN Hybrid Routing.

[6] Jun Li, Xiangqing Chang,Yongmao Ren, Zexin Zhang, Guodong Wang, 2014, An Effective Path Load Balancing Mechanism Based on SDN.

[7] Hyungbae Park, University of Missouri–Kansas City, 2015,High Availabilty and scalability schemes for Software-defined networks.

[8] High Availability for Non-stop Network Controller, Deguo Li, Li Ruan,Yinben Xia, Mingming Zhu, 2014.

[9] A Preliminary Research and Implementation of a Hierarchical High Availability Network Disaster-Tolerant System, Xiuqing Mao, Xingyuan Chen, Yingjie Yang,2011.