ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

Dynamic Load Balanced Packet Switching using Open Flow Protocol

Inderpreet Kaur[1], Abhishek Chakraborty[2], Bharat Bhadauria[2], Jaya Shree[2]

Asst. Professor, Dept. of CSE, GCET, Greater Noida, India[1]

Scholar, Dept. of CSE, GCET, Greater Noida, India[2]

ABSTRACT: The traditional network of TCP or UDP, make use of static switches, i.e. load balancing across paths which is derived by calculation of hash packets. Concerned issues with this approach are that each packet of such a flow goes in a solitary pre-characterized way through the network. In the case of any disparity in the path, either a switch breakdown or physical layer harm, bundles tend to lose or the other switches need to be manually configured for selecting a different path. This becomes a herculean undertaking, as the system develops. Likewise, the disservice of hashing is that all connections get the equal percentage of hash values or to say, all paths own the same capacity (ECMP - Equal Cost Multi-Path). Even if the network is coded strongly to work in as multipath networking, because of the identical capacity issue. Therefore, optimal load adjusting won't not be accomplished.
An optional answer for this issue is Software Defined Networking (SDN). SDN is an idea where the central controller takes the choice for parcel traversal and not the switches. Controller automatically distinguishes the topology by listening to the switches and ascertains path with lesser load. Controller then directs the switches with moving entries further needed for the paths thus optimally balancing the load with every flow.

KEYWORDS: TCP/UDP, static switches, equal-cost-multi-path, software defined networking, central controller, congestion control

I. INTRODUCTION

With the modernization and advancements in computer technology and availability of a number of distributed systems, the problem of load balancing within distributed systems has gained an imperative attention and importance. Consequently, a vast amount and variety of research have been going on in an attempt to solve the problem of load balancing.

With the introduction of distributed systems, it is becoming possible to obtain the maximum out of a set of computing nodes through a dynamic workload redistribution in order to avoid the situation where some of the hosts are idle while others have multiple jobs queued up. The impetus behind this load balancing involves two stages: efficiency and extensibility. The recent advances in computer and communication technology make a multi-computer method cheaper than a mainframe solution, with the same performance, provided all the computing resources are used effectively. The extensibility aspect of a distributed system should include options for the addition of new processors as the user needs arises. A well-designed load balancing algorithm aims at accommodating both the aspects. The ultimate goal of the design of such types of systems is to redistribute the global service demands created at different workstations over the dynamically available resources for computation.

Many algorithms have been reported in the literature. They differ in their performance objectives sought, the way of their data and control components, the characteristics of the system model used as a test bed, and the improving presumptions made to help the examination and the implementation of the simulation model. Under such models and assumptions, substantial mean reaction time upgrades at satisfactory expenses are reported effectively.

Modeling for execution concentrate on, the dispersed framework is ordinarily accepted to have homogeneous computing nodes and to be based on shared file server or every hub having its own particular neighborhood document framework. An optimistic view is mostly taken on some essential system attributes such as job transfer delays as pointed, load balancing overheads, and workload environment. No evaluation of framework outline options (e.g. correspondence conventions) has been reported till now.

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

To take accurate bits of knowledge into the execution of burden adjusting in disseminated systems, many numbers of realistic system characteristics need to be considered in terms of both load-balancing overheads and system model characteristics. We set out to check the legitimacy of the presumptions made on conveyed framework characteristics in other load balancing studies and to study the relative execution request of some regular burden adjusting calculations where more realistic models are assumed.

The remaining bit of this paper is sorted out as takes after. In Section II, we review the related work. In Section III, we describe the model and assumption. In Section IV, we present our proposed scheme. Section V shows the idea of load balance implementation and tools. In Section VI, we analyze our approach theoretically. Finally, we conclude the paper in section VII.

II. RELATED WORK

We propose two dynamic load balancing algorithms for multi-user jobs in heterogeneous distributed systems. The nodes in the distributed system are connected by a communication network. We consider two existing static load balancing strategy and extend them to dynamic schemes. These two existing load balancing strategies differ in their objective. (i) The first strategy, GOS [9] tries to minimize the expected response time of the system. (ii) The second technique, NCOOPC tries to diminish the normal response time of the singular clients (to give a client ideal arrangement). We base our dynamic strategy on the static strategy (i) and (ii). The performance of the dynamic strategy is compared with that of the static strategy using simulations with various system conditions. The results prove that, at low communication overheads, the dynamic strategy show superior performance over the static strategy. But as the overheads increase, the dynamic strategy as expected likely to perform same as that of the static strategy. The potential benefits of dynamic load redistribution to solve the occasional congestion experienced by a few nodes to improve the overall performance of the system are commonly accepted for distributed nodes. However, not even a single load balancing algorithm deals satisfactorily with the new and rapidly changing system conditions, and a dearth of up-to-date system state information. A load balancing algorithm comprises of two elements: information and control. The information element shares and keeps information about the state of the distributed system. Different techniques as to what information, how much information is to be maintained, how fast it is to be updated, and how large is the balancing district. 1 2 included, have been proposed. The heap file and their measuring techniques have also been the subject of a number of investigations. The control element uses this information to give decision when it is beneficial to redistribute the heap, who settles on this choice, which process to transfer, and where to transfer a process to reduce congestion and improve overall performance.

OpenFlow [7] is an unconventional networking technology that offers high interoperability and cheap ways of user control in data centered networks. It has been used for load balancing in a number of systems [13]–[15]. A typical OpenFlow network comprises of three components: the OpenFlow controller, OpenFlow switches, and the hosts. Each of the switches maintains a flow table that gives forwarding information. The controller and switches convey each other via OpenFlow messages. There is a series of actions that the OpenFlow controller is able to perform by sending messages to switches, like updating flow tables or probing switch statistics. By studying answering messages from switches, the OpenFlow controller can plan information flows effectively and efficiently.

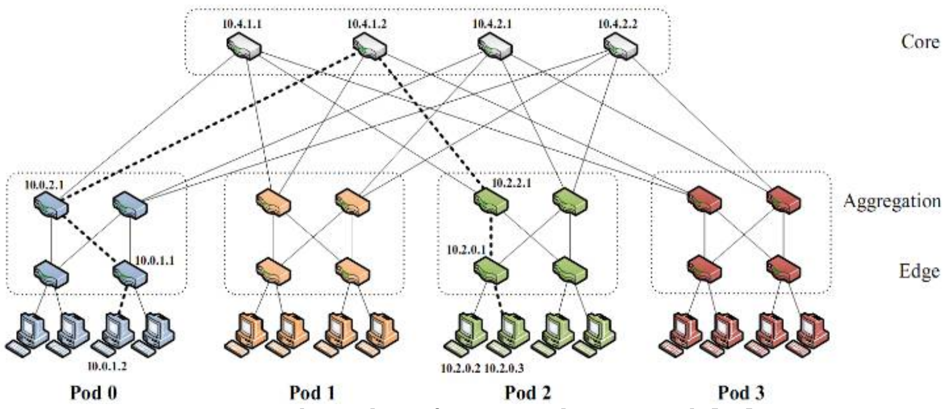

Fig. 1. A k=4 3-layer fat-tree topology network [15]

As a well-known server farm system topology, the fat-tree topology [2] have different ways among hosts so it can give higher accessible bandwidth in comparison to a single-path tree with the equal number of hubs. It is normally a 3-layer progressive tree that comprises of switches on the center, collection and top-of-rack (ToR) layers. The hosts make a connection to the switches on the ToR layer. The multipath highlight of fat-tree networks gives opportunities to convey information activity on various components of the network. It is a pragmatic approach to accomplish load-adjusting to schedule activity in fat-tree networks.

Symbol	*Description*
TCP	Transmission Control Protocol
UDP	Unified Datagram Protocol
ECMP	Equal Cost Multi-Path
SDN	Software Defined Network
ODL	OpenDaylight
DLB	Dynamic Load Balancing
SLB	Static Load Balancing
OF	OpenFlow
MAC	Media Access Control
SSL	Secure Socket Layer
SW/HW	Software/Hardware
IP	Internet Protocol
JVM	Java Virtual Machine
OSGi	Open Services Gateway interface
REST	Representational State Transfer
SAL	Service Abstraction Layer
DUT	Device Under Test
ToR	Top of Rack
ARP	Address Resolution Protocol

Table 1: List of Abbreviations Used

Distributed dynamic load balancing algorithms are likely to generate more number of messages than non-distributed algorithms. This is because each node might need to interact with all other nodes present in the system in order to make its load balancing decisions. However advantage is that the failure of one or more nodes in the system will not lead to the whole operation of load balancing to halt; it degrades system performance partially. Although the majority of element burden adjusting calculations proposed in the writing are distributed, yet it does not mean that the conveyed control is powerful in every one of them. For such algorithms that require each node to share status information with each and every other hub in the system, appropriated control could be a heavy burden on the communication system which affects the overall system performance adversely. Distributed control is of the great advantage when each node is given the maximum number of chances to act individually or to interact with as few nodes as possible. Quite imperative to say, most proposed dynamic burden adjusting calculations require full interaction among nodes of the distributed system. Hence, there is a huge clamor for distributed dynamic load balancing algorithms that call for minimum interaction among nodes. In a non-disseminated plan, the obligation of burden balancing is taken either by a single or a few nodes but never with all hubs. Non-conveyed based element load balancing can take two forms: centralized and semi-distributed. In a centralized form, the load balancing algorithm is executed by only one node of the distributed system: the central node. The central node is individually responsible for load balancing of the whole distributed system. Other nodes in the distributed system may react with the central node but not with each other. The interaction between nodes to achieve load balancing can take two forms either cooperative or non-cooperative. In a cooperative form, nodes work in harmony with each other to achieve a global objective, which is to improve the system's overall response time. In a non-cooperative structure, every hub works freely toward a nearby objective, e.g., to improve a local task's response time. Distributed dynamic load balancing algorithms tend to produce more messages than non-distributed algorithms. This is because each node might need to interact with other nodes in the system in order to perform its load balancing decisions. An advantage is that the failure of any nodes in the system will not cause the whole operation of load balancing to halt; it just incompletely debases framework execution. In spite of the fact that the majority of dynamic load balancing algorithms given in the literature are distributed, yet it does not mean that the distributed control is effective in all of them.

A dynamic load adjusting calculation is required to make load dispersion decisions based on the current status at each node of the disseminated framework. Therefore, this calculation must give a strategy for collecting furthermore, overseeing framework status data. The part of an element load balancing responsible for retrieving information about nodes in the system is referred as information methodology in the writing. Additionally, a dynamic burden adjusting algorithm must include a technique to assist each node in making the decision which job is eligible for load adjusting. The part of a dynamic burden adjusting calculation which chooses a vocation for exchange from a nearby hub to a remote hub is known as transfer strategy. In addition, a dynamic load adjusting calculation must give a strategy on which a destination center point for a traded occupation is resolved. The part of a dynamic load balancing algorithm that determines a destination node for a transferred task is referred to as area procedure. Hence, a dynamic burden adjusting calculation contains three main components: the information, exchange, and area systems. Each of these procedures will be described further. As shown in Fig. 2, incoming jobs are detected by the transfer strategy which decides whether or not it have to be transferred to a remote node with the end goal of burden adjusting. In the event that the exchange procedure gives a decision that a job should be relocated, the location strategy is triggered in order to find out a remote node for the employment. Data methodology gives both exchange and location schemes with the necessary information to build their decisions.

III. Model and Assumptions

Starting with, any network makes sense only if it is effectively connected, so that, each end nodes can communicate to all flip side hubs. In an exchanged fabric — a system topology that uses switches — the main objective is to connect a huge number of endpoints (processors or servers) by utilizing switches that only have a few number of ports. By skillfully interfacing exchanging components and framing a topology, a system can interconnect an amazing measure of endpoints.

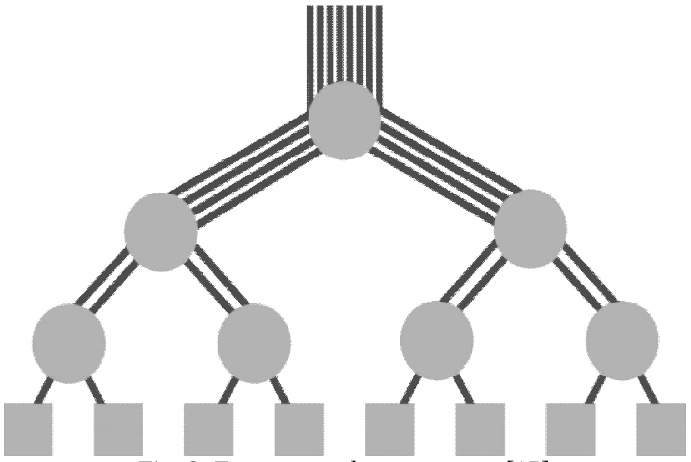
Fig. 2. Fat-tree topology structure [15]

Mininet is a system emulator. It runs an accumulation of end-hosts, switches, switches, and connects on a solitary Linux kernel. A Mininet host carries on simply like a genuine machine; you can ssh into it (if you start up ssh and bridge the system to your host) and run self-assertive projects. The projects you run can send packets through what seems like a real Ethernet interface, with a given connection speed and postpone. Parcels get handled by what resembles a genuine Ethernet switch, switch, or middlebox, with an idea of lining.

The physical division of the system control plane from the sending plane, where a control plane is in control of several other devices, is what is called a Software Defined networking (SDN). SDN is a technique to computer networking which evolved from work done at UC Berkeley and Stanford University around 2008. It is an evolving

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

engineering that is alterable, sensible, financially savvy, and adaptable, making it ideal for the high-data transfer capacity, dynamic nature of today's applications. This engineering decouples the system that gives decisions about where movement is sent (the control plane) from the hidden frameworks that forwards traffic to the selected destination (the data plane), consequently empowering the system control to end up straightforwardly programmable what's more, the fundamental framework to be dreamy for applications and framework organizations. The OpenFlow convention is a foundational element for building SDN solutions.

Iperf is a commonly-used network testing/emulation tool that is used to make TCP and UDP info streams and measure the throughput of a network that is transferring them. It can calculate the bandwidth and the quality of a network link. Iperf has a client and server functionality, and can calculate the throughput between the two ends, either unidirectional or bidirectional as depicted in Fig 3. DUT represents Device Under Test. It is open source programming and keeps running on different stages including Linux, UNIX, and Windows.

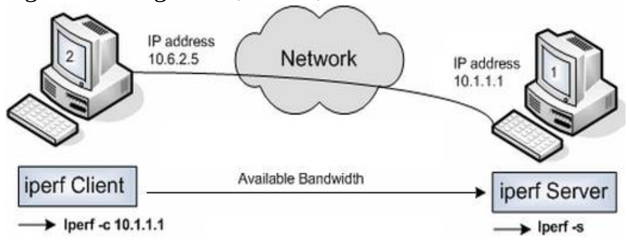

Fig. 3. Iperf measuring data bandwidth [12]

IV. Proposed Scheme

In this section, we propose to use the fat-tree topology, which involves multiple paths among hosts so it can provide higher bandwidth as compared to a single-path tree with the same number of nodes. It is typically a 3-layer hierarchical tree that comprises of switches on the center, conglomeration and top-of-rack (ToR) layers. The hosts interface with the switches on the ToR surface. The multipath highlight of fat-tree systems gives chances to distribute data traffic on different network components. It is a pragmatic approach to achieve load-adjusting to calendar movement in fat-tree systems.

Algorithm 1: DLB routing algorithm

```
Require: openFlow, fatTree
1: sourceHost = locateSource(flow);
2: destHost = locateDest(flow);
3: layer = setTopLayer();
4: curSwtch = locateCurrentSwitch();
5: direction = 1; //search in upward
6: path = null; //list switches
7: return pathSearch();
```

The DLB calculation plans flows based on current status and network statistics. Algorithm 1 [1] shows the pseudo code of our DLB calculation. This calculation functions as takes after. At the point when the OpenFlow controller receives a packet from a switch, it routes the control to the heap balancer. Line 1 to 6 presents the instatement for important variables. The load balancer initially checks the bundle's match data including the information port on the switch that obtains the packet as well as the bundle's source location and destination address. Then it checks those addresses using its knowledge about the system topology. Once the source and destination nodes are located, the load balancer retrieves the top layer that the flow needs to get to. We utilize the hunt bearing flag. The flag has two values: 1 for upward and 0 for downward. It is introduced to 1. A way is made for sparing a course grouped by a list of switches later. Line 7 calls search() that performs the search for paths recursively.

Algorithm 2: Recursive search for paths

```
1: pathSearch() {
2: curPath.add(curSwtch);
3: if isBottomLayer(curSwtch) then
4: return curPath;
5: end - if
6: if curSwtch.getLayer()==layer then
7: direction = 0; //search downward
8: end - if
9: linksList = findLinks(curSwtch, direction);
10: link = findWorstFitLink(linksList);
11: curSwtch = findNextSwitch(link);
12: return pathSearch();
13: }
```

Algorithm 2 describes [1] the method search(). It initially adds the current switch into the path. It gives the path back if current seek achieves the base layer. It inverts the inquiry direction if current search goes to the top layer. Then it calls for a technique that returns all links on the current switch that are towards ebb and flow look course. One and only connection is selected by picking up the worst-fit link with most extreme accessible data transfer capacity. And afterward the present switch article is overhauled. The technique search() is called recursively layer by layer from the source to destination. Finally, the path will be sent back to the load balancer. The path data will be utilized for overhauling flow tables of those switches in the path.

Algorithm 3: How to solve the problem

```
Control Loop {
Floyd Warshall Algorithm
Adjust links/connections
Run SPF for affected source-destination pairs
}
Stats monitor loop {
Read statistics for all nodes.
Maintain link utilization as a measure of total bandwidth of the link
Trigger Control Loop if link usage of any one link exceeds threshold
}
```

V. Open Flow-based Load Balance Implementation

As a module of the OpenFlow controller program, the heap balancer has two most important functionalities: monitoring ports measurements and booking new flows. It keeps up redesigned port statistics and schedules flows using our designed DLB calculation. Next, we outline them one by one.

1) Ports Statistics Monitoring: As shown over, the most exceedingly bad fit join determination depends on finding the link with the most available bandwidth. But an important question is how to measure the accessible data exchange limit on every association. It is difficult to gauge them in straight ways from the controller or switches. In our outline, the controller questions the transferred bytes on every port from switches intermittently, computes their increments and uses the increments as the major selecting criteria. It works as mentioned. We initiate by define a monitoring cycle as

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

time T. At the start of each monitor cycle T, the controller will convey a STATS REQUEST message doled out "PORT" as its measurement sort to each switch in the network. This message requires responses from the switches. In the interim, the controller keeps up an arrangement of ports S and the quantity of bytes they have sent over the last screen cycle as Tx. Each tuple of S comprises of three components as {switch_ID, port-number, Tx} format. When a switch answers to the controller with a STATS REPLY message, the controller will redesign Tx for the greater part of its ports in S according to corresponding {switch_ID, port-number} pair. The load balancer can find the worst-fit link by comparing Tx of all other links. The overhead of monitoring port statistics is nominal. The sizes of messages for requesting and replying port statistics on each port should be 8 bytes and 104 bytes respectively [11].

2) Flow Scheduling: The Flow scheduling functionality works as mentioned. Each OpenFlow switch maintains its own flow-table. Whenever any packet comes in, the switch checks the packet's match information with the entries present in its flow table. The packet's matching information includes ingressPort, etherType, srcMac, dstMac, vlanID, srcIP, dstIP, IPprotocol, TCP/UDPsrcPort, TCP/UDPdstPort [12]. If it finds a match, it will send out the packets to the corresponding ports. Otherwise, it will encapsulate the packet in a PACKET IN message and forward the message to the controller. As a module of the OpenFlow controller, the heap balancer will handle the PACKET IN message packet. It finds a suitable way by executing an inquiry with the DLB calculation depicted in Algorithm 1. The way is a rundown of changes from source to the destination of the packet required. Then the load balancer creates one FLOW MOD message packet for each switch in the way and sends it to the switch. This message will have the packet's matcher information as well as an output port number on that corresponding switch. The output port number is calculated by the way and system topology. On the off chance that one switch gets a FLOW MOD message, it will use it to update its flow table suitably. Those packets buffered on ports of that switch should find their matches in the updated flow table and be conveyed. Otherwise, the switch will rehash this procedure as long as required.

VI. Evaluation and Results

In this section of our paper, we describe the evaluation environment, traffic outline and estimation approaches. We conduct tests with Mininet on Linux host machine with the OpenDaylight OpenFlow controller. Then we measure the network throughput and dormancy under various sorts of traffic examples, such as via iperf.

Top-of-Rack switches have some initial and final number of packets transmitted through them. The final values should concur with the initial values in the sense that no one switch gets overloaded with packets, while the other one just sits idle.

The following data is observed from the execution of our test cases:

Node Connector ID	Rcvd Pkts	Sent Pkts	Rcvd Bytes	Sent Bytes
openflow:10:1	54	3	8890	261
openflow:10:2	27	3	4848	261

Table 2:Intial Data for Core Switch 1

Node Connector ID	Rcvd Pkts	Sent Pkts	Rcvd Bytes	Sent Bytes
openflow:20:1	60	34	10617	5552
openflow:20:2	53	40	9200	6862

Table 3:Intial Data for Core Switch 2

The final values are important for observation. They are given as follows:

Node Connector ID	Rcvd Pkts	Sent Pkts	Rcvd Bytes	Sent Bytes
openflow:10:1	267	11	32808	957
openflow:10:2	39	11	6442	957

Table 4: Final Data for Core Switch 1

Node Connector ID	Rcvd Pkts	Sent Pkts	Rcvd Bytes	Sent Bytes
openflow:20:1	309	257	44084	34196
openflow:20:2	276	289	37844	40329

Table 5: Final Data for Core Switch 2

As can be observed from the data above, when the openflow:20 switch started to overload, the switch openflow:10 came to the rescue and finished off with that switch.

As a result of the above observation, we can see that our algorithm concurs with our hypothesis and hence is considered successful.

VII. CONCLUSION AND FUTURE WORK

In this paper, we exhibit a dynamic burden adjusting calculation to efficiently schedule the flows for fat-tree networks, which give numerous option ways among a solitary pair of host machines. The algorithm utilizes the hierarchical feature of fat-tree systems to recursively hunt down a way and makes decisions based on real-time traffic statistics obtained through the OpenFlow protocol. We have implemented the algorithm as a module of the OpenDaylight OpenFlow controller program, with two fundamental capacities: observing traffic insights and planning flows. In conjunction with the OpenDaylight controller, we utilize the Mininet system emulator to assess the dynamic load balancing algorithm, by comparing it with the no-load-balancing and static-load-balancing algorithms. The results show that our algorithm is better over the other two in maintaining a high rate of data transmission and avoiding network latency under various sorts of framework traffic issues. In our future work, we plan to amplify the dynamic burden adjusting calculation to traditional TCP/UDP networks with only regular switches, then again half breed systems with both OpenFlow and consistent switches.

REFERENCES

1. Li, Y., & Pan, D. (2012, November) OpenFlow-based Load Balancing for FatTree Networks with Multipath Support.
2. van der Pol, R., Boele, S., Dijkstra, F., Barczyk, A., van Malenstein, G., Chen, J. H., &Mambretti, J. (2012, November). Multipathing with MPTCP and OpenFlow. In High-Performance Computing, Networking, Storage, and Analysis (SCC), 2012 SC Companion: (pp. 1617-1624). IEEE.
3. Nguyen, S. C., Zhang, X., Nguyen, T. M. T., &Pujolle, G. (2011, May). Evaluation of throughput optimization and load sharing of multipath TCP in heterogeneous networks. In Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on (pp. 1-5). IEEE.
4. Wang, R., Butnariu, D., & Rexford, J. (2011, March). OpenFlow-based server load balancing gone wild. In Proceedings of the 11th USENIX conference on hot topics in the management of internet, cloud, and enterprise networks and services (pp. 12-12). USENIX Association.
5. W. J. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publisher, 2004.
6. S. Kandula, S. Sengupta, A. Greenberg, P. Patel and R. Chaiken. The Nature of Data Center Traffic: Measurements & Analysis. ACM IMC 2009.
7. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM CCR, 2008.
8. R. N. Mysore, A. Pamporis, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A Scalable, Fault-Tolerant Layer 2 Data Center Network Fabric. ACM SIGCOMM, 2009.
9. Beacon OpenFlow Controller. https://OpenFlow.stanford.edu/display/Beacon/Home.
10. OpenFlow Switch Specification, Version 1.0.0. http://www.OpenFlow.org/documents/OpenFlow-spec-v1.0.0.pdf.
11. http://en.wikipedia.org/wiki/Software-defined_networking
12. https://www.opennetworking.org/index.php?option=com_content&view=article& id=686&Itemid=272&lang=en
13. http://jackbrowntelecomprofessional.wordpress.com/2011/10/19/software-define d-networking-sdn/
14. http://image.slidesharecdn.com/20140224-definitionsv2-140305082348-phpapp0 2/95/acronym-soup-nfv-sdn-ovn-and-vnf-6-638.jpg?cb=1394110989
15. http://clusterdesign.org/fat-trees/fat_tree_varying_ports/
16. http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm