Load Balancing and Congestion Control in Software Defined Networking using the Extended Johnson Algorithm for Data Centre

M. Bindhu, Dr. Ramesh, G.P.
1Research Scholar, Department of ECE, St. Peter's University, Chennai, India
2Professor, Department of ECE, St. Peter's University, Chennai, India
1deepbind@gmail.com, 2rameshgp@yahoo.com

Abstract
Software-Defined Networking (SDN), showcases a paradigm shift from traditional network towards a new era of Internet. Distributed controllers addresses hindrances like availability and scalability which fails to provide a dynamic mechanism for load balancing. Implementation of Dynamic load balancing safe guard the network from sudden traffic congestion which could be created by data spikes or link breakdown. This paper propose different attempts to integrate congestion control (CC) and dynamic load balancing by combining SDN approach along with Johnson Algorithm. The Extended Johnson Algorithm takes into consideration not only edge weights but also node weights with negative cycles. The algorithm produces efficient utilization of network resources, high throughput of all pairs with shortest paths in a sparse weighted, dense and directed graph. Implementation of this proposed algorithms is studied with OMNET emulation tool under the Abilene network topology.

Keywords: Software-defined networking (SDN), Congestion Control (CC), Load Balancing, Data-Centre.

1. Introduction
Software Defined Networking (SDN) technology separates the control plane and data plane of network devices [1]. In SDN, a logically centralized controller configures (add/delete) the forwarding tables (also called flow tables) of switches rather than MAC address tables, which are responsible for forwarding the packets of communication flows. This secure channel requires very small bandwidth compared with the other channels of the network that connect routers [1]. To monitor and manage the entire network in a centralized and real-time manner, SDN users can fuse application programs run on top of the controller.

In this scenario, each location has its own OpenFlow controller which has Ethernet switch behavior. The OpenFlow switches are analysed to buffer the packets and send the header fields to the connected controller. The entire network uses one end host per OpenFlow domain with Ethernet links. On the start-up, switch initiates a secured channel with controller over TCP, which is managed by OpenFlow Protocol. The switch knows the required IP address of the controller by configuration. The controller is also possible to detect the switch and initiates the connection setup.

Either way is possible, the switch and controller exchange hello messages to learn the highest, once the secure TCP connection between switch and controller is done. OpenFlow version is supported by both entities. The OpenFlow version supported by both is found, the controller requests for all capabilities of connecting switch through a features request message. The receiver switch answers the featured request message with a featured reply message and thereby informs the controller about its supported functionality.

Because of larger information the router sends more packets than allowed rate through one port. The dropping of extra packets will result in loss of information. Open Flow protocol[2] uses a Controller which can manage the traffic across the network for better load balance trying to reduce the channel congestion. At present, network traffic is growing fast and complex as enterprises need to purchase more equipments to handle this complex network. The serious problems faced by enterprises is Network congestion and server overload. This problem can be resolved using smart load balancing technique[3] among routers and channels, where the controller efficiently manages with full knowledge about the topology of all the components in the network. The network administrators control the switch programmatically which reduces the manual configurations on switch, with greater flexibility in network management.

The traditional load balancing products cannot be used in data centre with virtualisation environment. Many business are transforming into cloud environment where the virtualisation technology help companies like cloud computing, big data in integration of servers with maximum utility[4], thereby saving money.

This paper says about load balancing algorithm which helps in finding the shortest path using extended Johnson algorithm for SDN based networks under OMNET emulation tool[5] under the Abilene network topology[6]. The rest of this paper is organized as follows: Section 2 briefs about the related works in SDN Load balancing, Section 3 describes Proposed work algorithm, Section 4 discusses about the simulated results and finally section 5 concludes with possible improvements and continuation in the work.

2. Related work
SDN based Load Balancing
The online services like e-commerce, websites, social networks frequently use multiple servers to get high reliability and accessibility. The paper [7] proposed a load balancing algorithm, named LABERIO (LoAdBalanceEd Routing with OpenFlow), minimizes latency and response time increases the network throughput. The algorithm uses ToR (Top of Rack) Switch-to-ToR Switch Paths Table (S2SPT) and Load Allocation Table (LAT). But if the network topology changes S2SPT becomes a problem. So, LABERIO is not suitable in wide area network because in the wide area we cannot predict the topology changes. The paper [9] proposed the Plug-in-Serve system implementing a load balancing algorithm, called LOBUS (LOad-Balancing over
UnStructure networks)[8], using OpenFlow for unstructured networks, yields the lowest total response time for each newly arriving request. The figure 1 shows about SDN based load balancing diagram in general. The routers receive the flow packets from the hosts along with request sends to controller. The flow packet has details of source, destination, data rate and priority. The controller supervises used/unused bandwidth with data rate of all links and decides on:

1. No congestion on the network, decision of the controller is to order the router (where the packet is originate) to send the flow to same path.
2. Congestion occurs at any link, the new flow is processed or change data rate flow through routing algorithm.

IP services use Open Shortest Path First(OSPF), the computation is based on Dijkstra's algorithm which calculates the shortest path within the network. The major disadvantage of the algorithm is a blind search is done which involves in time consumption. Secondly it cannot handle negative edges.

The proposed Algorithm can be applied to derive all pairs shortest paths, in a sparse weighted, directed graph with negative numbers. This algorithm is faster than Floyd–Warshall on sparse graphs. Johnson’s algorithm uses as subroutines both the Bellman-Ford algorithm an edge from u to v, having length w(u,v), is given the new length. Dijkstra's algorithm is used to find the shortest paths from each node to every other vertex in the reweighted graph. To prevent network congestion, the idea is to send each request to the nearest server with the link load lower than a pre-specified threshold \( \theta \). Though, if all the servers have link loads larger than the threshold, the algorithm still wish the nearest server. The idea of Johnson’s algorithm is to re-weight all edges and make them all positive, then apply Dijkstra’s algorithm for every vertex. One may think of a simple approach of finding the minimum weight edge and adding this weight to all edges. Unfortunately, this doesn’t work as there may be different number of edges in different paths. If there are multiple paths from a vertex u to v, then all paths must be increased by same amount, so that the shortest path remain the shortest in the transformed graph. The idea of Johnson’s algorithm is to assign a weight to every vertex. We reweight edges using vertex weights. The great thing about this reweighting is, all set of paths between any two vertices are increased by same amount and all negative weights become non-negative.

3. Proposed Extended Johnson Algorithm

The algorithm solves all pairs shortest paths, in a sparse weighted, directed graph with negative numbers. This algorithm is faster than Floyd–Warshall on sparse graphs. Johnson’s algorithm uses as subroutines both the Bellman-Ford algorithm an edge from u to v, having length \( w(u,v) \), is given the new length. Dijkstra's algorithm is used to find the shortest paths from each node to every other vertex in the reweighted graph. To prevent network congestion, the idea is to send each request to the nearest server with the link load lower than a pre-specified threshold \( \theta \). Though, if all the servers have link loads larger than the threshold, the algorithm still wish the nearest server. The idea of Johnson’s algorithm is to assign a weight to every vertex. We reweight edges using vertex weights. The great thing about this reweighting is, all set of paths between any two vertices are increased by same amount and all negative weights become non-negative.

3.1 Time Analysis: The time complexity of this algorithm, using Fibonacci heaps in the implementation of Dijkstra's algorithm[9], is \( O(V^2 \log V + VE) \); the algorithm uses \( O(VE) \) time for the Bellman–Ford stage of the algorithm, and \( O(V \log V + E) \) for each of \( V \) instantiations of Dijkstra's algorithm. Thus, when the graph is sparse, the total time can be faster than the Floyd–Warshall algorithm, which solves the same problem in time \( O(V^3) \)[10].

The node weight \( nw[v] \) of \( v \) is defined according to Eq. (1),

\[
nw[v] = \sum_{e \in \text{flow_bits}(v)} \frac{1}{\text{Capacity}(v)}
\]

and the edge weight \( ew[e] \) of \( e \) is defined according to Eq. (2).

\[
ew[e] = \sum_{e \in \text{flow_bits}(e)} \frac{1}{\text{Capacity}(e)}
\]

The technique of reweighting Assign new weights \( \hat{w} \) to each edge as follows:

\[
\hat{w}(u, v) = w(u, v) + d(s, u) - d(s, v)
\]

\[
w(u, v) + d(s, u) - d(s, v) \geq 0
\]

Reweight the edges so that[10] [11]

1. No edge weight is negative.
2. A shortest path in the original graph is still one in the new, reweighted graph.

Figure 1. SDN load balancer environment
Johnson’s algorithm
1. Compute \( G' \), which consists of \( G \) augmented with 
   \( s \) and a zero-weight edge from \( s \) to every vertex in
   \( G \).
2. Run Bellman-Ford(\( G' \), \( w \), \( s \)) to obtain the \( d(s,v) \)’s
3. Reweight by computing \( \hat{w} \) for each edge
4. Run Dijkstra on each vertex to compute
5. Undo reweighting factors to compute \( d \)
7. \( d[v] \leftarrow d[u]+ew[u,v]+nw[u] \)
8. \( p[v] \leftarrow d[u] \)
9. if \( p=\emptyset \) then
10. \( s \leftarrow \min(P) \). server
11. else
12. \( s \leftarrow \min(Q) \). server
13. return \( s \)

Run time:
1. Computing \( G' \): \( Q(V) \)
2. Bellman-Ford: \( Q(VE) \)
3. Reweighting: \( Q(E) \)
4. Running (Modified) Dijkstra: \( \Theta(V^2 lg V + VE lg V) \)
5. Adjusting distances: \( Q(V^2) \)
6. Total is dominated by Dijkstra: \( \Theta(V^2 lg V + VE lg V) \)

4. Simulation
4.1. Simulation software: The paper consists of an Abilene 
   network topology, can be implemented with Ethernet, fibre or 
   wireless connections. The network modelling software 
   OMNeT++[12] simulation is used to build the project. 
   OMNeT++ is an object oriented simulation that depends on 
   both C++ and Network Description (NED) languages. The 
   OpenFlow protocol uses the OpenFlow switch with INET 
   framework [13]. For the implementation, the openflow.h header 
   file is created to model the protocol and its specified 
   messages. The implemented nodes consists of OpenFlow 
   switch and OpenFlow controller along with important 
   messages for the communication between switch and 
   controller on the OpenFlow channel. In addition to the 
   OpenFlow nodes, other utility modules are installed which 
   provide further required functionality like a spanning tree 
   module and a controller placement module. In this Abilene 
   topology only a single OpenFlow controller is dynamically 
   placed in the network which connects to the different 
   OpenFlow switches.

4.2 Results:
The Abilene core topology, set up has an OpenFlow Controller 
with 5 OpenFlow switches as SDN nodes, each of which was 
linked to the controller logically. Each domain consist of an 
OpenFlow switch, several hosts and optionally an OpenFlow 
controller. The number of hosts can be defined by the user, as 
shown in Fig 2.
Ipref command is used for TCP data stream from client to server. For the request size $2^{10}$, the response size is 65536 bytes. The figure 5 and figure 6 shows, the latency is calculated by ping command by sending packets for 30 seconds. The flow of packets using openflow switches responds very well with uniform packet distribution at times of 4 nano seconds. Also the openflow switch 5 uses Ethernet MAC at maximum data transfer. In real data centre networks, IPRouters, controllers will add have minimum of certain microseconds added as latency link. The timing changes with respect to farthest server with lots of switches. The time taken by the client to get reply command from the server is Netpref which calculates response time, its calculated by sending request size of $2^{10}$ for 30 seconds.

This paper described a load balancing method in virtual environment based on OpenFlow technology which solves the problem of how to take load balancing into network virtualization in data centre. The paper also explains, that it is possible to make a powerful, flexible and cost effective load balancing methods by OpenFlow. OpenFlow method grants flexibility for the realization of different load balancing strategies, which is conveniently used in software-defined network to achieve different load balancing strategies in different network environment. The algorithm results in efficient utilization of network resources with high throughput in shortest paths identification in a sparse weighted, dense and directed graph.

5. Conclusion

References