Scalability Issues in Hierarchical SDN Control plane

Suraj kumar Jha

Scalability Issues in Hierarchical SDN Control plane

Report submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

by

Suraj kumar Jha
(140101074)

under the supervision of

Professor Sukumar Nandi

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, INDIA
November, 2017

DECLARATION

I declare that

1. The work contained in this thesis is original and has been done by myself and the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
4. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT Guwahati
Date:

Suraj kumar Jha
Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Guwahati, INDIA 781039

CERTIFICATE

This is to certify that the work contained in this thesis entitled "Scalability Issues in Hierarchical SDN Control plane" is a bonafide work of Suraj kumar Jha(Roll No. 140101074), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.

Place: IIT Guwahati
Date:

Professor Sukumar Nandi
Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Guwahati, INDIA 781039

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern network is evolving rapidly due to increase in Internet users and diversity in application demands. Therefore, the traditional way of network management is inadequate for managing complex integrated control and data plane architecture driven network. "Software Defined Networking" (SDN)[1] is a paradigm which uses separation of control and data plane to make the network more manageable. Thus, SDN helps in the rapid evolution of the network.

1.1 Software Defined Networking

Although SDN has been defined in many ways, in this report we adapt to the definition provided in [1] and has the features as follows.

1. SDN separates data and control plane. "Control plane" of a network deals with how to route the packets. On the other hand, "Data plane" of a network deals with the forwarding of packets based decision taken by the control plane. All data plane devices become mere forwarder of packets based upon the prescribed rules by the control plane.

2. A flow is defined as a set of packets with matching header attributes. In SDN, the forwarding decision is made per-flow basis rather than per packet basis. The forwarding decisions associated with flows are called action set. Packets of same flow undergo identical treatment. Flow helps in aggregating packets so that better routing decisions could be made by the control plane.

3. Packet forwarding criteria is generated by "SDN controller" or "network operating system" (NOS) based upon logically centralized and abstract view of the network. Controller creates forwarding rules and sends them to the data plane devices.

4. The control plane algorithm could be changed at SDN controller without affecting any network nodes. This feature makes the network programmable.

Fig.1.1 shows different components of SDN. Forwarding elements of network for example switches and routers reside at Infrastructure layer. They have flow tables installed with matching criteria ranging from Ethernet to TCP/IP header fields. Switches match these criteria to drop, forward and rewrite header contents of the packets. A flow table miss is informed to control layer which in turn generates and installs the matching criteria on switches. Control layer can also install flow proactively. Control layer consults "Application layer", which runs business logic while generating matching rules. The flow tables also store flow level statistics, which can be requested

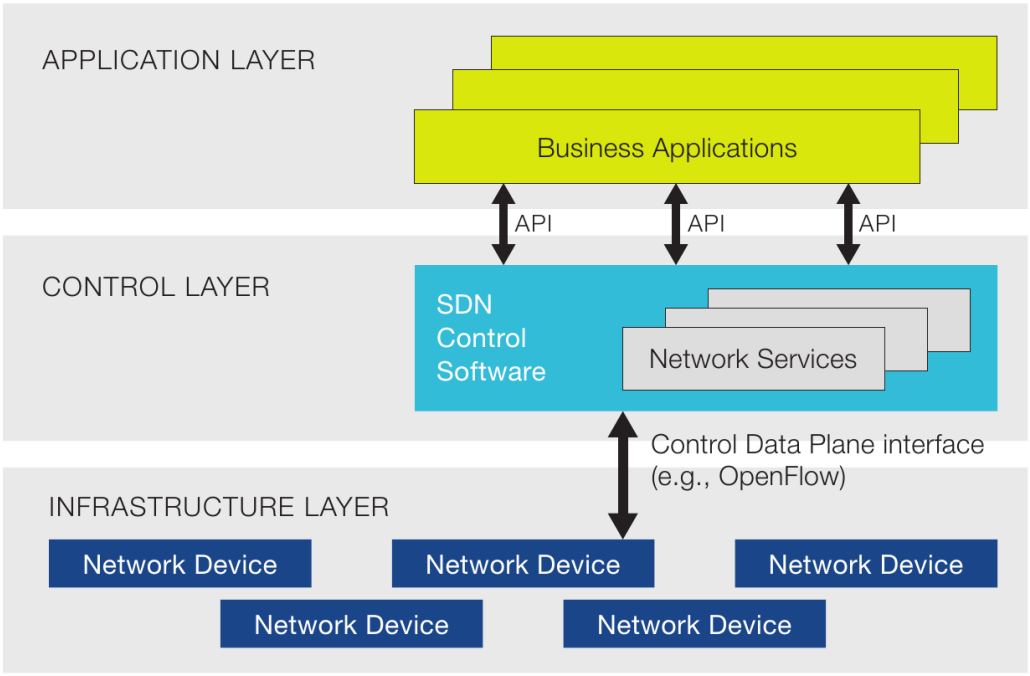

Fig. 1.1 SDN: Architecture

by control layer to be passed on to the application layer that may further analyze the working of the network.

Software defined networking has many features to offer. The cost will come down as commodity hardware can be used instead of expensive network equipment. In the traditional network, each device has its own vendor and device specific CLIs, configuration files and GUIs. So in a heterogenous Network, admins would need to go to each and every device to make changes. This is very cumbersome as well as error-prone. But in SDN, devices can be controlled centrally. Programmability of network makes it easy to experiment on new protocols. Centralization brings the real-time visibility of the network, while programmability can be used to take necessary actions rapidly. This would be very helpful in the case of security threat, to ensure higher uptime, better performance and overall a hassle-free and controlled network.

It would be better to get an idea of the different type of large-scale network before looking into scalibility issues. We take two different scenarios to understand the difference: Datacenter versus ISP. Datacenters contain very large number of devices which are spatially closely located and there is low and predictable delays between links. The flow requirements can also be estimated very precisely. In such cases, flow can be setup proactively. In ISP the number of devices is lower than that of datacenters but the devices are spatially distributed. Devices have high and unpredictable latencies also the traffic requirements at any time could not be predicted accurately, we can not set up flow proactively in this case. The proactive flow set up reduces communication between controller and switches and thus makes network scalable. In the absence of proactive behavior, even with a smaller number of devices, for example ISP the scalability gets tough.

In today's complex network, control plane is not only expected to provide end-to-end connectivity but also has to satisfy the new set of demands that include traffic engineering, access

control, isolation, and virtualization. These different goals increases the importance as well as load on control plane devices, which brings the issue of how to implement SDN in a large-scale network.

1.2 Scalibility issues in SDN

In SDN, the first packet of each new flow is sent to SDN controller. Controller would generate flow table entries for the switches. In a big network, this requires a lot of communication between control and data plane devices, which presents a major scalability issue. For example In a large-scale network wherever the controller is placed the latency to setup a new path will be high.

Different approaches are proposed to solve this problem. One approach suggests logically centralized but physically distributed control plane. Distributed control plane divides the whole set of network devices amongst controllers who take local decisions minimizing latency in decision making. Each controller contains network-wide state and so gives logically centralized view. The main limitation comes when the network-wide state has too much information (in a big network) to handle and so scalability is limited.

In contrast to that Hierarchical control plane creates an abstract view of the controller. Hierarchical control plane has different type of controllers, one in bottom layer which does not have global view but can take decisions within its domain and other in top layer which has logically centralized view and so run applications which need network-wide state like routing. Bottom controller thus handles most of the local traffic generated from frequent events e.g Link Layer Discovery Protocol and protects top layer from frequent events.

In hierarchical control plane, top layer controllers abstract bottom layer controllers as big virtual switches, with links connecting two controller domain as virtual links. This abstraction helps in running application like routing which needs the global view of the network. Due to abstraction, there are two types of links one is interdomain and another is intradomain link. Intradomain links connect two switches under same local controller, while interdomain link connects two switches residing in different controller domain. Setting up network path between hosts which lie in different regions needs global network view and so top-layer controller aka super controller decides the path and passes this information to local controller. The super controller can only inform the interdomain link to local controller, but the actual path setup would also involve nodes inside a local controllers domain.

Since super controller does not look into link statistics of intradomain links while choosing the path, It is quite possible that the path chosen by super-controller for interdomain communication may not be optimal and may not satisfy the Quality of Service (QoS) criteria of the flow. e.g A bottleneck link inside a controller domain may cause reduced throughput for interdomain communication if the bottleneck link is part of the path chosen.

1.3 Organization of The Report

This chapter provides a brief introduction to SDN and scalability issues in SDN. Chapter 2 summarizes how different approaches and previous solutions tackle scalibility issues in SDN. Chapter 3 explains about our implementation of a hierarchical control plane. It is used to test the validity of proposed problem and will also be used to evaluate the solution. Chapter 4 shows detail analysis of an example scenario where the proposed problem exists. Chapter 5 explains about the approach towards solution and gives mathematical formulation to the problem. Chap-

ter 6 describes about experiment scenario and implementation details. We present our results and conclusions in chapter 7. Chapter 8 describes future work which may improve upon the solution presented in this report.

Chapter 2

Review of Prior Works

There have been several attempts to make SDN scalable and more manageable. Imroving controller"s processing capabilities, reducing requests to controllers and deploying multiple controllers are some of the approaches taken to make network scalable and manageable. While many network nodes can be easily added to the distributed control plane architecture, the overhead in managing a global network view increases as each controller have only a partial view. Any change in controller level would require all the controllers to participate. In contrast to that hierarchical controllers are more easy to manage as upper layers abstract out lower layers as big virtual switches connecting with virtual links. Any new change at upper layers could be easily made without affecting lower layer devices but this poses scalability issue. If the upper layer controllers are consulted too frequently and for events that could be processed locally, a lot of resources would be wasted and latency would increase.

2.1 Distributed control plane

Hyperflow [2] and Onix [3] are example of distributed control plane, i.e physically distributed but logical centralized. Controllers in Hyperflow publish network-wide states to other controllers to keep a consistent global network view. Onix provides two data stores one is Network Information Base (NIB) where applications write to and read from network-wide information, the other is memory-based one-hop Distributed Hashtable (DHT), which is fast and would be eventually consistent. This gives applications choice between speed and durability.

2.2 Hierarchical control plane

Google [4] has global wide SDN deployment for interconnecting data centers which manages their traffic engineering services centrally. Each datacenter has a site controller like local controller, also for fault tolerance there is a selection of site controller from several software instances available at different physical servers. Site controllers are managed by global controller. To avoid scalability issue, they proactively establish flows and gather flow statistics less frequently and for a small number of flows. They can establish flow proactively [5] Kandoo separates events and thereby applications into two types: local and global event (applications) and does not consult upper layer controller for frequent local events like link discovery with LLDP. This minimizes the overhead at upper layers and helps in scaling. While Kandoo suggests two different controller types,[6] RAON proposes recursive and hierarchical control plane with a single controller type. RAON recursively divides network and then abstracts for the upper layer as a big virtual

switch connected with virtual interfaces. Leaf layer controllers control switches while upper layer controls their immediate children controllers.

2.3 Hybrid control plane

ORION [7] is an example of hybrid control plane which believes that neither hierarchical nor distributed control plane will be able to resolve all the issues. Distributed architecture suffers from scalibility issue while hierarchical one introduces path-stretch problem (the ratio between the actual path taken and shortest path between any two nodes increases). ORION proposes three-layered hybrid control plane in which there is a hierarchy in bottom layer devices, but the top level controllers run a distributed protocol to share network-wide informations.

2.4 Closure

In all cases of hierarchical control plane implementations, we find that the upper layers take the decision by considering abstract views of bottom layers, ignoring the lower level link details. Hence it is possible that the routing decisions taken by the upper layer may be non-optimal.

Chapter 3

Implementation for problem verification

OpenFlow [8] is one of the first and most popular Software defined networking Standards. It defines the communication protocol between SDN controller and forwarding plane devices like Switches and routers. We have used Mininet [9] and Ryu [10] for the experiment. Mininet creates a realistic virtual network by running real kernel, switch and application code, on a single machine (VM, cloud or native). It is also used to develop Open FLow and SDN systems. With the help of Mininet one can create virtual open flow switches and controllers. Ryu is a python based software defined networking framework which supports various protocols including OpenFlow. Ryu provides events and APIs to communicate with openflow switches. We have used Ryu remote controller and mininet to create virtual hosts and switches to implement hierarchical control plane.

3.1 Hierarchical control plane details

We have developed upon the base Hierarchical control plane code obtained from [11]. The local controllers(L_C) are ryu controllers and super controller (S_C) is a python server to whom local controllers(L_C) connect via socket. Both local and global controller asynchronously publishes events to each other. Tables in annexure lists main events published by S_C as well as by L_C. These events facilitate route setup, switch, host status and topology information passing between controllers.

The existing code needs topology information at S_C level via a configuration file. The configuration file lists information about links and switches containing different controller domains. That makes it a static topology Hierarchical control plane. It can not accommodate any change in topology at runtime. But to test different scenario for our problem, we require a control plane that can dynamically discover the topology and could respond to changes. Ryu has topology discovery modules which provides some basic sets of asynchronous events which informs local controller about different events like changes in topology due to host or switch connected or left, but these changes are informed at L_C level and contain the information within a controller domain. To achieve a network wide state, these information needs to be passed to S_C. Table 6.3 ,6.4 and 6.5 lists Ryu and Openflow events and Ryu APIs whuch are used by Local controller to get topology information. But the topology discovery in this way can not be used to discover inter-domain links (links connecting two switches residing in different controller domain). So we have made some changes in Ryu code to discover inter-domain link.

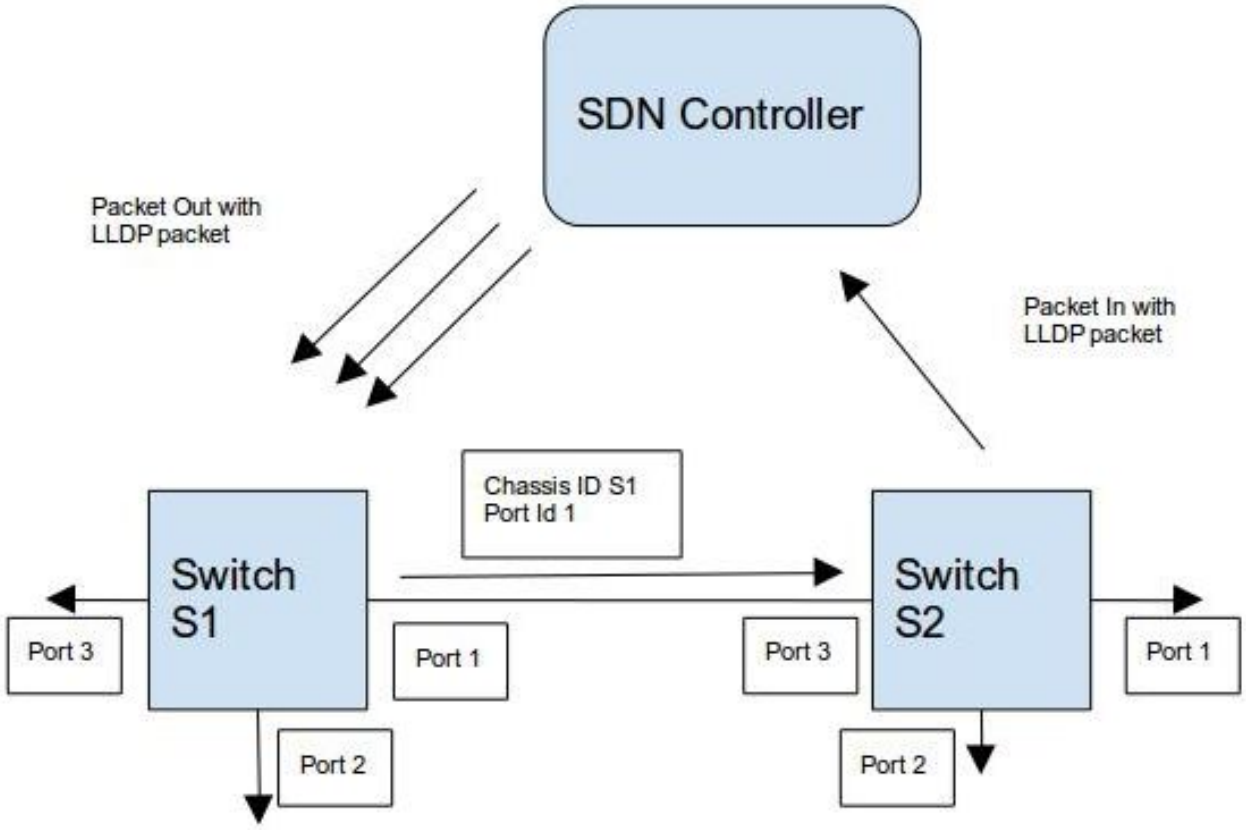

Fig. 3.1 Link discovery in Ryu

3.2 Link discovery in ryu

To discover a link between two switches, LLDP packets are sent. The controller sends LLDP packet to each of the port of the switch. LLDP packet contains the information about the source switch and port number, from which the packet has originated. These packets are addressed to a bridge-filtered multicast address and so not forwarded by switches. After one hop these packets are sent back to SDN controller which in turn looks into the source Switch id and port and Destination switch id and port, where the packet was received. From the above example, we can see one such movement of LLDP packet, this information conveys there is a link between Switch S1 port 1 to Switch S2 port 3. [12]

But by this way, a link between two switches cannot be discovered if they lie in different controller domain. Take the example of below topology S3 and S4 lies in different controller domain C1 and C2 and so above mechanism would fail because LLDP packet sent by C1 to S3 when forwarded to S4 and then received by C2 would be discarded as the packet would not be acknowledged by C2.

This behavior can be modified to discover inter-domain link. The local controller when presented with such cases does not discover these links directly but also not rejects these packets.They keep these thinking that these packets may contain inter-domain link information.

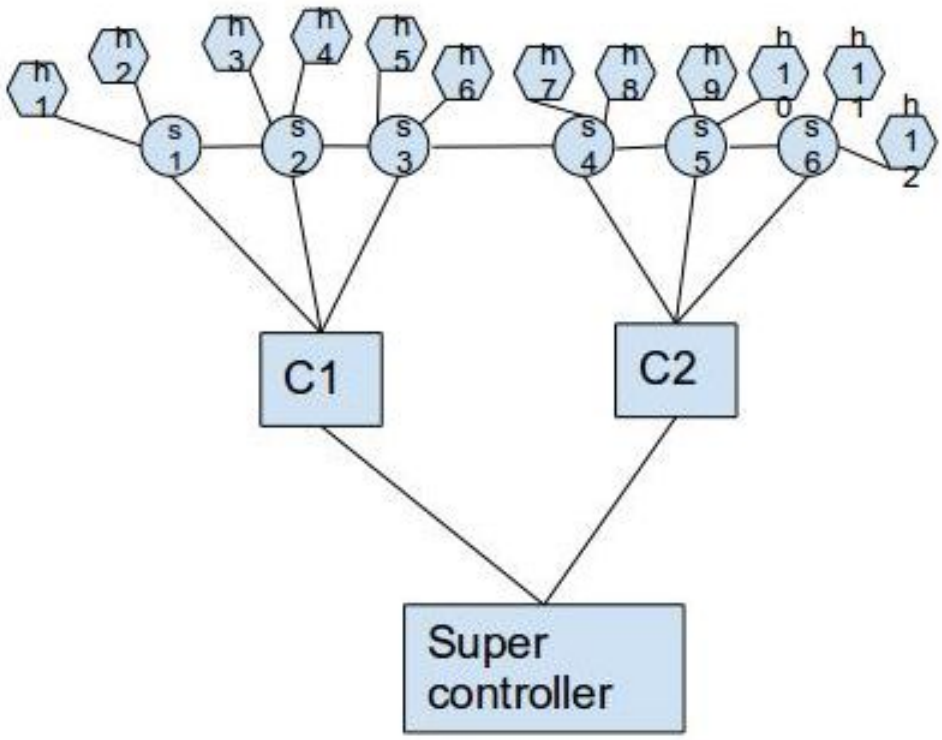

Fig. 3.2 Link discovery between two controller domains

The information is passed to global controller periodically or when it is perceived that there is a topology change. S_C receives the link information between S3 and S4 from both C1 and C2 controller.Thus adds a link between C1 and C2. The link properties specify the port number and other parameters for routing.

For example, LLDP packet sent by S3 when received by S4 will have src_dpid : s4, src_portno : 2, dst_dpid : s3 , dst_portno : 1. C2 after receiving this would pass to S_C, which adds an edge to the toplogy graph connecting C1 with C2 having edge property right_dpid : s4, right_portno : 2, left_dpid : s3 , left_portno : 1. Right is for destination controller and left for src controller. There can be several paths between two controllers. All such paths are added.

Chapter 4

Example scenario

Here we look into one example scenario in which due to lack of information at super controller level, non-optimal routing decision is made.

4.1 Problem Setting

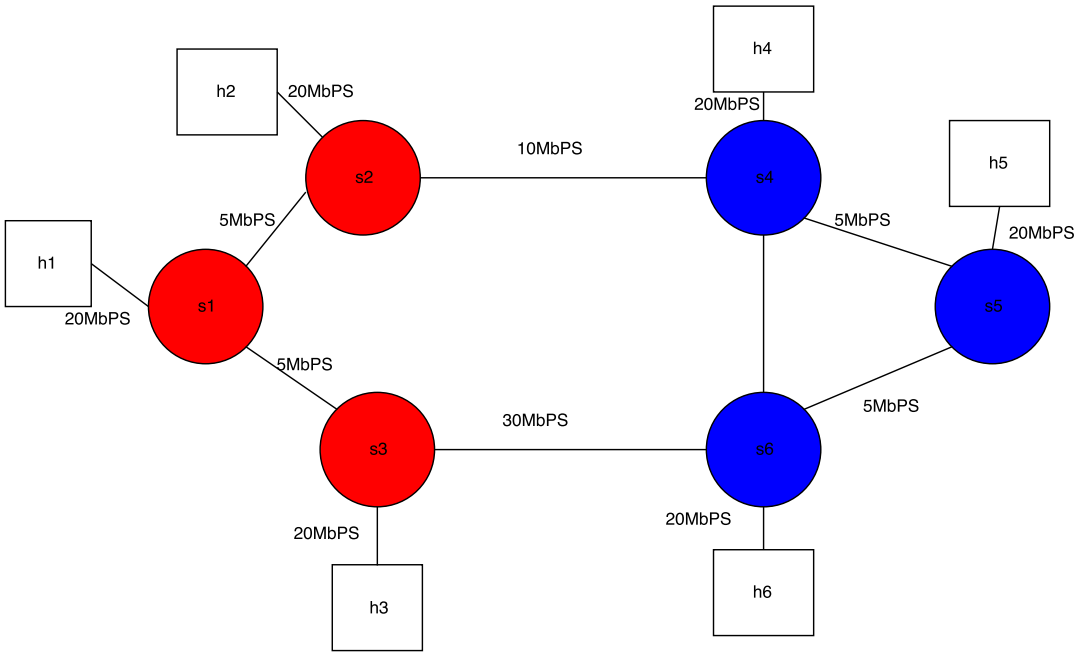

Fig. 4.1 Problem scenario

Table 4.1 Traffic already present

Host-node pair	Path	flow Bandwitdh requirement
h1-h6	path: h1-s1-s3-s6-h6	4MbPS
h3-h5	path: h3-s3-s6-s5-h5	4MbPS

Switches in red which are S1, S2 and S3 are controlled by Controller C1 and in blue which are S4, S5 and S6 are controlled by C2. Servers are running on h1 and h3. h6,h5 and h4 are

Table 4.2 Available bandwith at inter domain links

Link	Available Bandwidth
s2-s4	10MbPS
s3-s6	22MbPS

clients. Currently, h6 and h5 are connected with h1 and h3 respectively. Now let us suppose, h4 tries to connect with h1 with flow requiring a bandwidth of 4MbPS. h1 is under C1's domain while h4 is in C2,hence an inter-domain path setup is required. S_C is consulted, S_C looks into the available bandwidth of interdomain links, because that is the information it will have. There are two paths available from C1 to C2 via s2-s4 and s3-s6. The better path as per available bandwidth is through s3-s6. So S_C says L_C to set path from s1 to s3 and C2 to set path from s6 to s4 for route setup. Since the link s1-s3 is already carrying one traffic the path chosen is not optimal. Here s3-s6 would become bottleneck link. The pptimal solution would have been to choose s2-s4 as interdomain link, though that is not the best path in the view of S_C, considering available bandwidth as criteria. Here Routing decision taken at S_C level in the absence of link level details forces L_Cs to set up a non optimal path.

4.2 Performance measurement

We have run two type of test one using Iperf [13], other by creating HTTP server at h1 and h3 and requesting using wget from h1, h6 and h4. In one scenario S_C chooses the non-optimal path whereas in other case, it chooses optimal path for routing. We have measured and compared different network parameters for the optimal and non-optimal scenario.

4.2.1 TCP Throughput

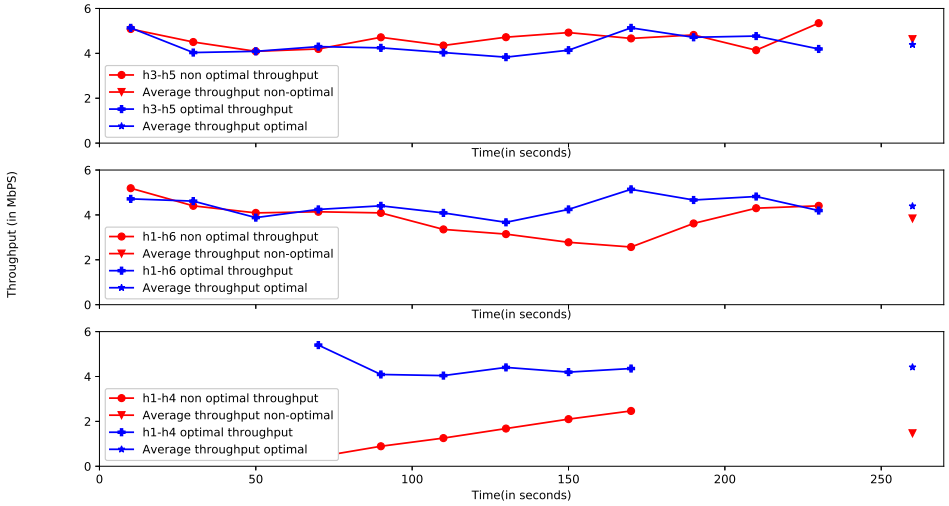

Fig. 4.2 TCP throughput test Iperf

Iperf tcp server is created at h1 and h3. h6 and h5 connects with h1 and h3 at t=0, h4

connects at t=60s. We find that till t=60, there is not much difference between throughput of the optimal and non-optimal case. When h4 tries to connect, it gets very poor bandwidth (less than 1MbPS). s1-s3 link is congested. TCP at h4 finds this out and does not increase its congestion window fastly, which leads to the poor throughput of h1-h4 connection. As the time passes, packet loss occurs for h1-h6 flow also and so TCP running at h6 also reduces its congestion window. At that point, the bandwidth starts getting shared. Thus the total bandwidth is shared between h1-h6 and h1-h4 flow. The average bandwidth for h1-h6 in the non-optimal case is less than that of optimal, but in h1-h4 case this gap is huge.

4.2.2 UDP Throughput

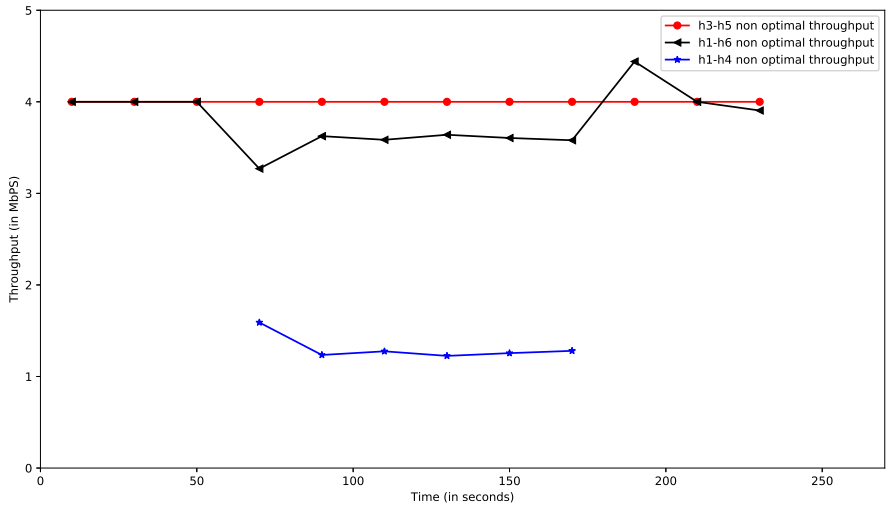

Fig. 4.3 UDP throughput test Iperf

The same scenario as above is created. UDP servers are running at h1 and h3 instead of TCP. The target bandwidth for each connection is set to 4MbPS. The graph shows non-optimal case as in optimal case all three traffic gets 4MbPS. Due to s1-s3 bottleneck link datagram loss occurs leading to less throughput for h1-h4, making way for h1-h6 traffic. Due to lack of congestion control mechanism of UDP, all UDP clients are still sending datagram at the rate of 4MbPS, even if there is packet loss. Both h1-h4 and h1-h6 traffic suffers. Bandwidth is shared between h1-h4 and h1-h6 traffic. Unlike TCP case here h1-h4 traffic throughput does not change much over the course of the experiment. h3-h5 traffic remains unaffected, as it does not have any botlleneck link its path.

4.2.3 Datagram loss

We have analyzed the datagram loss in case of UDP. We find that h1-h4 traffic suffers hugely in this case. On an average it loses around 70% packets, due to congestion. This is the cause of reduced throughput of h1-h4 flow in case of UDP. h1-h6 also suffers the loss of datagrams but the loss is very less as compared to that of h1-h4. In optimal case, there is no loss of datagrams in any of the flows.

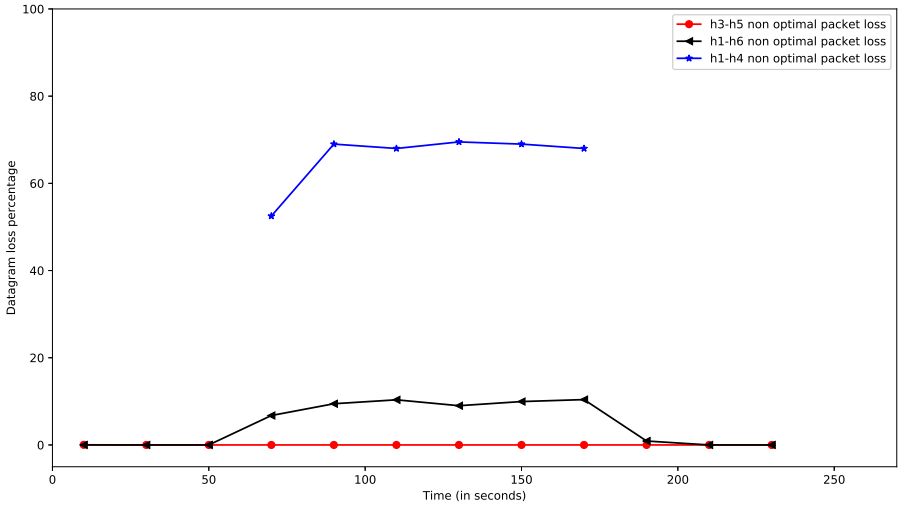

Fig. 4.4 Datagram loss percentage test

4.2.4 Average RTT

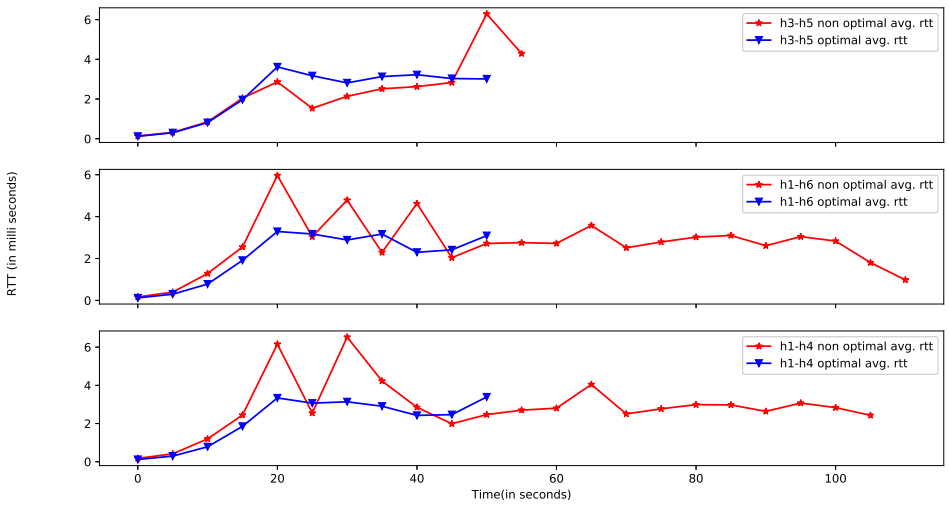

Fig. 4.5 Average RTT test HTTP server

We have used HTTP server - wget to analyze the behavior of RTT in both the cases. Simple Python multiconnection servers are created at h1 and h3. h4, h6 and h5 request a file of size 25Mb using wget. Here also we find that h1-h4 and h1-h6 traffic suffers, their throughput decreases and RTT increases. The time taken to download the file in optimal case is almost half of that in the non-optimal case for h1-h6 and h1-h4, indicating that average throughput is double in the case of optimal path selection than that of non-optimal path. h3-h5 traffic is not

much affected. RTT increase is caused due to s1-s3 bottleneck link.

4.3 Analysis

Measurement and comparison of different parameters show that A bottleneck link in the chosen path may cause severe loss in throughput, increased round trip time, high increase in packet loss percentage. In short, almost all the network parameters get affected which may lead to the sub-optimal performance of the network. The severity of effect varies from parameter to parameter and also with the protocol like TCP or UDP. But it is evident that a wide range of applications would get affected due to this.

Chapter 5

Solution approach

As we have seen in the last chapter that super controller can make bad decisions for flow routing as it does not and can't have all the link level informations related to the ongoing traffic. But at the same time local controller could decide about flows based upon more details as it may track link properties. It would be much better to have lesser number of flows that need super-controller's consultation. The idea is to have a clustering methodology which divides switches into different controller domains, such that total number of inter-domain flows are minimized. Also if there are more number of interdomain flows, super controller would be more busy handling the requests, which may be a concern as super controller may become bottleneck. So the clustering of switches which gives lesser number of interdomain flow updates to super controller is better. It is quite possible that to satisfy this criteria, some controllers may be assigned more number of switches than they can handle. So the criteria of load balance also should be brought. A heavily loaded local controller would not be able to satisfy the needs of all flows in its domain.

5.1 Mathematical Formulation

Given the data plane of a data center network G = {N, E}. Switches (**N**) are controlled by out-of-band hierarchical control plane. Let, **C** be the set of leaf level controllers. We call switches associated to same controllers as switches of same domain/cluster. All intra-domain flows can be setup without super controller's help. Each inter-domain flow setup requires consultation with the super controller as the inter-controller links are controlled by super controller. Let N be the set of switches and C be the set of clusters. Let C_i be the ith cluster. An element of C_i is S, where S is one switch. Let F be list of flows F_1, F_2, ..., F_o which is about to arrive in next time quantum say δ. For a flow F_i, which passes through an ordered set of nodes N_1, N_2 , .. N_j. f(C,F) represents number of inter-cluster flow table updates required. Our objective is to minimize f(C,F), when we donot have control over F. f(C,F) is our cost function.

$$\text{f(C,F)} = \sum_{i=1}^{i=o} \sum_{k=1}^{k=j-1} \text{g}(N_y^k)$$

where, $\text{g}(N_y^k) = 1$ if N_y^k and N_{y+1}^k does not belong to same domain else 0

Objective : Minimize f(C,f) with
Constraint 1 : Each switch should be exactly matched with one and only one controller.
We define size of a cluster as number of switches in its domain.
Constraint 2: $\forall$k, (1/k-1/ak) $\leq$ Size(C_k) / Size(C) $\leq$ (1/k+1/ak)

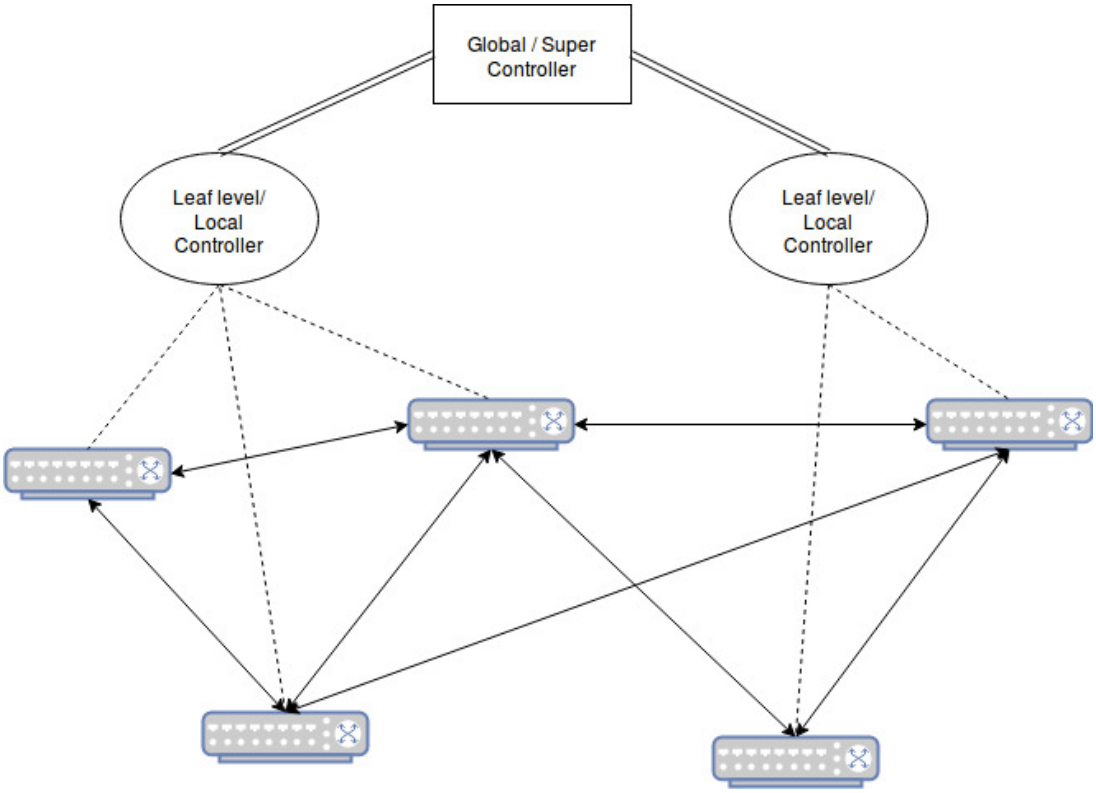

Fig. 5.1 Hierarchical SDN structure
Double arrowed line represnts physical link
Dotted line represents switch controlled by leaf level controller
Doubled line with no arrow repesents connection between leaf level controller and super controller

where a is load relaxation factor. if a tends to ∞ then each controller domain would get equal number of switches in its control.

Constraint 3: All the switches inside a controller domain should be physically connected.

The rational behind this constraint is to that if all the switches of a controller domain is not connected then to setup a flow between its own pair of hosts, the consultation of Super controller would be required, which if true, does not match with our assumption that all intra domain flows should be handled by Local controllers.

5.2 How to solve this optimization problem

Our objective function does not have a known mathematical representation, neither we do know about the convexity or derivative of this function. Also evaluating the objective function is expensive. As for example in our case to evaluate a given clustering, we need to run the traffic and then evaluate the objective function. **Bayesian optimization** is a powerful strategy for finding the extrema of such objective functions that are expensive to evaluate. It is applicable where objective functions does not have a closed form but one can obtain observations of the function at different possible samples. It is particularly very useful when evaluation of such function is costly and the derivates of the function are not known.

Although cost function is unknown, it is assumed that there exists some prior knowledge about some of its properties, which makes some objective functions more possible than others. Bayesian optimization also tries to find a tradeoff between exploration (where the objective function is very uncertaion) and exploitation(where the objective function is expected to be high). Bayesian optimization technique has a nice property that it aims to minimize the number of objective function evaluations.

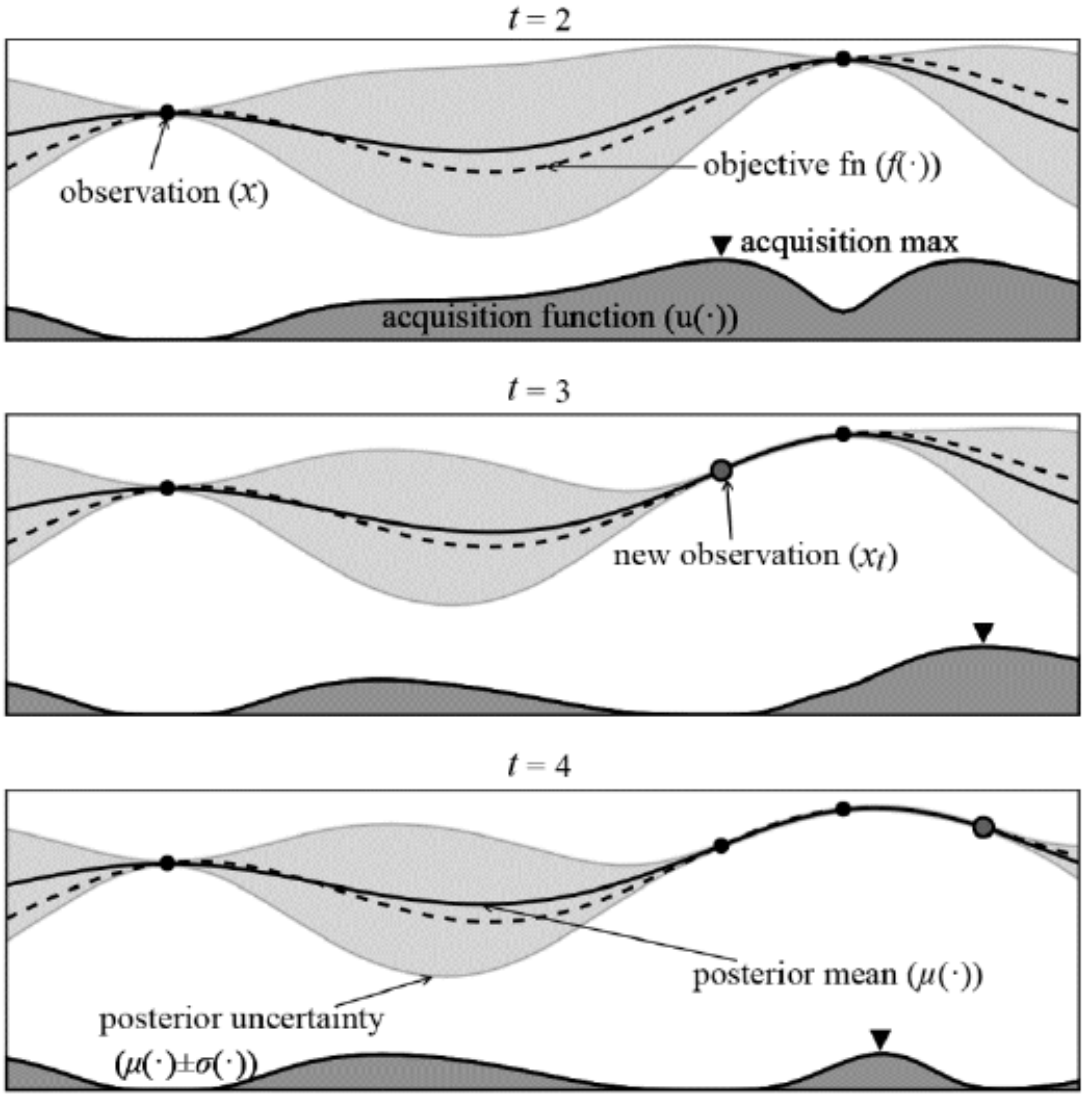

Fig. 5.2 Sample 1D Bayesian Optimization iterations

5.3 Bayesian Optimization

Bayesian optimiation uses a prior and evidence to define a posterior distribution over the space of functions. Which next point to choose for sampling is decided by a utility function called acquisition function. The next point sampled is evaluated and again added to the existing dataset.

Bayesian optimization tries to fit a Gaussian process to our observed data points and

pick next best point where it belives the maximum will be. A guassian process is an extension of the multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution. Guassian process can be thought of analogous to a function where a function returns a scalar, Guassian process returns the mean and variance of normal distribution over the possible values of f at a sampled point x.

The next point to explore is decided by an acquistion function which trades of between exploitation and exploration. When exploring, we should choose points where the variance is large. When exploiting, we should choose points where the mean is high.

5.3.1 Choice of covariance function

Covariance function gives the smoothness to optimization function. It also helps in finding the similarity between two sampling points and so helps in choosing the next sampling point. A sample covariance function can be

$k(x_i,x_j) = \exp(-1/2\|x_i - x_j\|^2)$

where x_i,x_j are samped points. Different covariance function exists. Matern [14] is the covariance function we are using here

5.3.2 Choice of acquisition functions

Different acquisition functions available are:

- Probability Improvement : It tries to maximize the probability of improvement over currently optimal value. The drawback is that this formulation is more exploitative.
- Expected Improvement : This tries to take into account not only the probability of the improvement but also the magnitude of the improvement, thereby trying to maximize the expected improvement when choosing a new trial point.
- Lower confidence bound : This tries to reduce the domain of exploration by selectively rejecting areas where not much improvement could be expected.

Chapter 6

Experimental setup

In this section we decribe our experimental setup and different modules that we use to perform the experiment.

6.1 Topology

We have used clos-tree topology for the experiment. It is one of the widely used and popular data-center topology. In this topology, every lower-tier switch is connected to each of the top-tier siwtches in a full-mesh topology. This helps in creating a non-blocking architecture which has a larger number of paths between any two host. So even if some links are busy, due to some traffic going on, suitable path can be chosen. Another advantage of the clos tree network is that it can be built using a set of identical and inexpensive devices. It gives high gain in performance and resilience. If the path is randomly chosen, the traffic load would be equally distributed across top-tier switches. Even if one of the top-tier switches fail, it only degrades performance slightly.

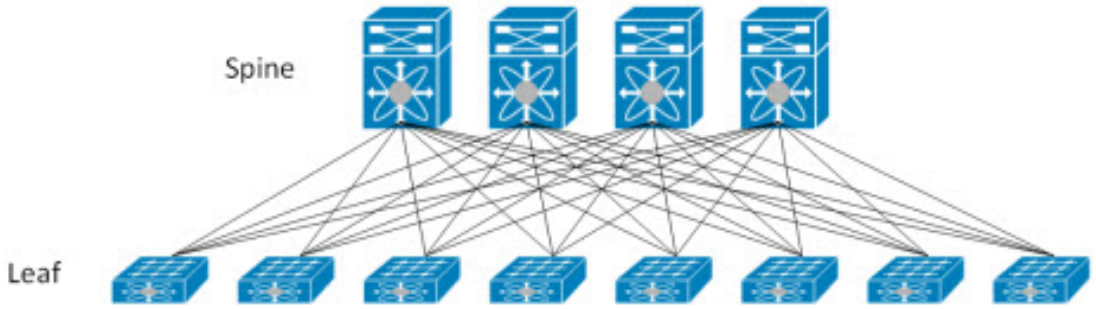

Fig. 6.1 Clos Tree topology

In this experiment we are using 3 spine switches and 6 leaf switches. For each leaf switch there are two hosts attached. So there are 12 hosts and 9 switches. To control the traffic we are using 3 leaf level controllers and one super controller. Leaf level controllers control spine and rack switches. Each leaf level controller can handle flows which has both source and destination under its domain independently without considering super controller. While for the flows which have origin and destination in different leaf level controller. Super controller is consulted which provides input to leaf level controller about how to setup path.

6.2 Data traffic characteristics

We use Open source datacenter traffic pattern obtained from Maxinet [15]. Each flow has four attributes

- source host
- destination host
- time of start of flow
- number of seconds flow will continue

All flows start within 60 seconds of start of traffic and have variable flow time. For this experiment we are only interested in knowing the number of super controller updates these flows cause. So we restrict the flow time for each flow to be 60 seconds. As once the path is established, however long the flow runs, it does not matter much for super controller, but with large flow time our evaluation function would take too long and so experimentation would be difficult. With this restriction, all flows should be finished with in two minutes time period. There are a lot of flow entries in the original datset which are not valid for us, because we have a limited number of hosts. So we filter out only those flows whose both source and destination hosts are present.

6.3 Experimentation flow

Steps of the experiment:

1. Create hosts, switches, links and controllers (both local and super) with the help of mininet[9] and ryu [10]
2. Run bayesian optimization module. which decides which configuration to sample next.
3. Connect switches to corresponding leaf level controller as per configuration
4. Run topology discovery module as described in section 3.2, because inter-domain links could have changed due to change in switch and leaf-level controller mapping.
5. Start traffic to evaluate objective cost funtion
6. Gather number of flow table updates from super controller.
7. Go to stage 2 for next iteration of bayesian optimization.

6.4 Switch migration

Switch migration is essential to implement dynamic change in switch-controller association. Open flow provides a way by which one can setup and change switch to controller.

Steps in switch migration:

1. After picking a configuration, bayesian optimization module informs super controller to enforce the switch-controller mapping.

2. Super controller informs each leaf-level controller to start switch migration.

3. Leaf level controller iterates through each of the switch connected and finds if the switch should still be connected to it or the switch is to be migrated to other controller.

    - For a switch which needs no migration, no action is taken
    - For a switch which needs migration, a RPC request is made to a server which can run open-flow command to set switch to the requested controller.

4. Each leaf level controller finishes this operation and informs super controller.

5. Super controller informs Bayesian optimization module, which may proceed further with experiment.

It would be worth noticing that the liveness guarantee of a switch at the time of migration is not guarrantted in the current implementation. But it does not harm our experiment in anyway as at this time there are no ongoing flows and no new flow can arrive. Had this not been the case we need to guarrantee safety and liveness property of the switch. For details of which one might refer to the switch migration module presented in elastic SDN [16]. The method proposed in the above paper is open-flow compatible and uses Barrier request and reply to implement it.

6.5 Bayesian optimization

We have used Bayesian optimization module of skopt [17]. Skopt is one of the popular and opensource optimization library implemented in python. For bayesian optimization, the implementation supports all major acquisition functions that is Expected improvement, Probabilty improvement and lower confidence bound. It also supports popular covariance matrices like Matern, which we are using here.

Chapter 7

Result and conclusion

7.1 Results

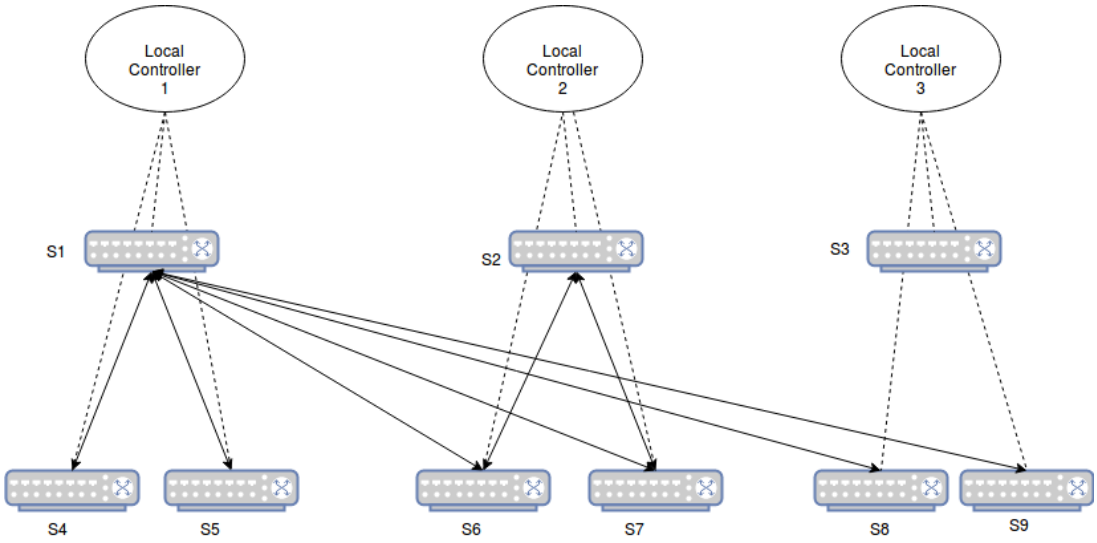

Fig. 7.1 Experiment topology and a sample configuration
Rack switches : s1,s2,s3 Spine switches : s4,s5,s6,s7,s8,s9 Like s1, s2 and s3 are connected with rack switches in mesh fashion.

We take a sample configuration in which each leaf-level controller controls one spine and two rack switches for our reference. The average number of inter-domain flow updates for this configuration is 880. We compare configurations chosen by bayesian optimization over iteration with respect to the above configuration.

We see that Expected Improvement(EI) acquision model performs better than Probability improvement and Lower confidence bound (LCB) performs the worst. The lowest possible value is obtained by EI (280). Best configuration picked is [1, 3, 3, 1, 2, 1, 1, 1, 3]. We performed only 10 iterations for LCB because for last 4 iterations, LCB was choosing the same configuration. The different values in the objective function evaluations for last 4 iterations is due to noise in function evaluation, which is explained in the next section. EI tries to maximize the expected value for getting minimum number of updates and so explores through different configuration which may have high probability of giving minimum value. Due to this the variation in values of

different configuration chosen is high for EI, whereas LCB does not choose a lot of configurations. Variation in its values is due to noise.

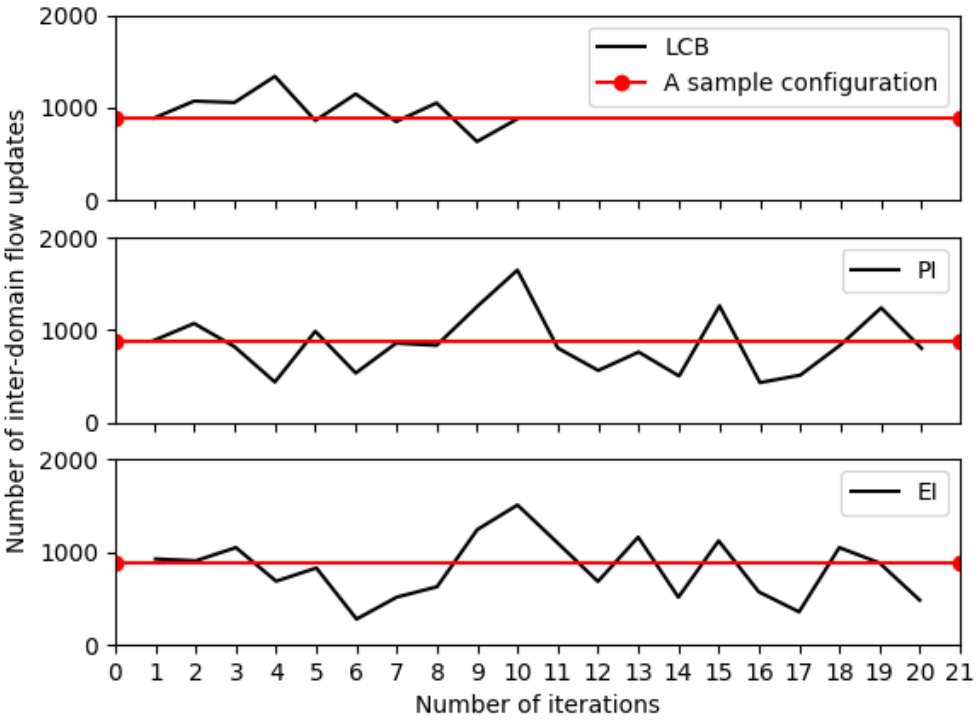

Fig. 7.2 Number of updates over iteration for Bayesian optimization
Red line denotes number of updates chosen for configuration decsribed above

7.2 Observation

We observe that though bayesian optimization is able to find some of the configurations which have very lesser number of flow updates value (280 updates for EI). Still none of the model seems to converge. Though LCB chooses one configuration more number of times, minimum number of updates achieved by this are not so good. A Bayesian optimization method will converge to the optimum if [18]

(i) the acquisition function is continuous and approximately minimizes the risk (defined as the expected deviation from the global minimum at a fixed point x); and

(ii) conditional variance converges to zero (or appropriate positive minimum value in the presence of noise) if and only if the distance to the nearest observation is zero.

We have observed that our cost function is noisy. That is even for the same configuration when applied across different runs, objective function can return different value. As for example take [3, 1, 1, 3, 3, 3, 1, 2, 3] configuration where ith entry specifies to which controller ith switch would connect to. As for example first switch would connect to 3rd controller. When this configuration is applied two different times, in one of the cases we get 887 updates, in another

Table 7.1 Number of updates vs runs

Number of run	Configuration	Number of updates
1	[3, 1, 1, 3, 3, 3, 1, 2, 3]	821
2	[3, 1, 1, 3, 3, 3, 1, 2, 3]	887

case we get 821 updates. This change in number of possible updates is possible. Assume that for an inter-domain flow the first request packet to super controller is generated, but super controller could not reply within the time period in which leaf controller received another request packet for the same flow. In this case there may be more than one requests per flow, Which cause changes in the number of updates as perceived by super controller.

Another point worth considering is the effect of covariance function on bayesian optimization. Though bayesian optimization does not need to know the details about black box function. It need to have some sort of mechanism to say which configurations are more closer to optimal than others. Based upon this closeness value the bayesian optimization chooses to pick the configuration which is closer to optimal value. In current implementation we have used Matern kernel. Matern kernel is based upon the physical distance between two configurations. Take a

Table 7.2 Geometric vs actual similairity

Configuration1	Configuration2	Geometrical similarity	Actual similarity
[3, 1, 1, 3, 3, 3, 1, 2, 3]	[3, 1, 1, 3, 3, 3, 1, 2, 3]	very high	very low
[1, 1, 1, 1, 1, 1, 1, 1, 1]	[3, 3, 3, 3, 3, 3, 3, 3, 3]	very low	very high

configuration [1, 1, 1, 1, 1, 1, 1, 1, 1] and other configuration [3, 3, 3, 3, 3, 3, 3, 3, 3], these two configuration are way seperate from each other because the physical distance between these two would be highest as compared to any other two configuration, where number of controllers are 3. But these two configurations are essentially identical. In both the cases all the switches are under a single controller domain and so for both the cases number of flow updates at super controller level would be 0. We examine another scenario from our experiment. The difference

Table 7.3 Number of updates vs configuration

Configuration1	Number of updates
[1, 3, 2, 3, 2, 2, 2, 3, 2]	433
[1, 3, 2, 3, 2, 2, 1, 3, 2]	1262

is configuration between these two configuration is very small i.e 7th switch changing from 2nd controller to 1st controller. This will be interpreted as a small distance by matern which is based on geometrical distance of two points. But it is quite possible that a significant number of flows now would become inter-domain flows, requiring significant number of inter-domain flow updates. This case is represented by the above scenario.

Since the convergence criteria is based upon the smoothness provided by covariance function, it can be concluded that bayesian optimization is not converging because the covariance function is not appropriate for this case. In next chapter we propose a covariance function which we may be better suited to provide similarity measure between two configurations.

Chapter 8

Future work

As observed in the last chapter, Matern covariance function, which uses geometric distance to give similarity measure, is not appropriate for the bayesian optimization problem in hand. If the covariance function could include traffic charectiristics and provide similarity measure based on that, bayesian optimization may converge and may be able to perform better. We propose a covariance function without giving experimental proof about the result based on this covariance function.

8.1 Flow based covariance function

Let us take two configurations CC1 and CC2. Suppose n_1,n_2,...n_k be the nodes which changed their controller from configuration CC1 to CC2. For a vertex v, let x be the number of inter-cluster edges in the shortest path between n_i and v in CC1 and y be the number of inter-cluster edges in the shortest path between n_i and v in CC2. Let the probability of communication between between n_i and v be P_i. Then change between CC1 and CC2 due to this link is $(P_i * \|y - x\|)$. We compute the change due to the node n_i by summing the change for all pair of nodes between n_i and v, where v is any other node. This gives change due to one migrating node from CC1 to CC2, we similarly compute for all k migrating nodes and add them to get the similarity value between two clusters.

Mathematically,

Covariance(CC1,CC2) = $\sum_{i=1}^{i=k}$ Q(N_i,CC1,CC2)

where,

Q(N_i,CC1,CC2) = $\sum_{j=1}^{j=\|v\|}$ P(N_i,j) * (Change in intercluster edges between N_i and j from CC1 to CC2)

P(N_i,j) = Number of flows between N_i and j / Total number of flows in the system.

This covariance function gives weightage to the flows existing between two hosts, as the probabiltiy of flows between them would be higher. So the configuration, in which two highly communicating nodes would be placed under a single controller domain would have high dissimilarity than the configuration, in which these two nodes are placed under different domains. We analyse the case taken by us in the last chapter.

Configuration 1 : [1, 1, 1, 1, 1, 1, 1, 1, 1]

Configuration 2 : [3, 3, 3, 3, 3, 3, 3, 3, 3]

From Configuration 1 to Configuration 2, all switches migrated from controller 1 to controller 3. But in both the configuration all flows would be intradomain, because in first case all switches are in Controller 1 domain and second case all switches are in Controller domain

2. So change in intercluster edges between both the configurations is 0 and hence Covariance(CC1,CC2) would be 0. This matches with the actual similarity value. It also comes in contrast with geometrical distance percieved by Matern kernel.

Similarly, we can take other example when a small change in configuration (migration of one switch as shown in last chapter) can produce a large change in actual similarity, if the migrating switch produces lots of inter-cluster flows. So it would have larger share in change of number of inter-cluster edges.

We believe that this covariance measure provides a better similarity value between two configurations and so would help in converging Bayesian optimization. We also believe that this could produce better configurations, which has less number of inter-cluster updates.

At the same time it would be worth noting that the time complexity to evaluate this covariance function would be higher than simple geometric covariance. Traffic flow characteristics also need to be logged during the time of experiment to calculate the probability of flow occurrence between two nodes. So though this function would be more costlier than Matern, it may be worth its cost in improving bayesian optimization.

Chapter 9

Annexure

Table 9.1 Different messages sent from local to global controller.

Message Type	Information	Reply message
HelloUp	Local controller is up,assigned an id	HelloDown
DPConnected	Switch connected to controller, its dpid is assigned and is added into topology graph	RoleAssign
GidRequest	Request for new datapathId, keeps on increasing when asked for	GidReply
PacketIn	Request for routing to the destination not under the domain of local controller.	FlowMod
LoadReport	Load report published by controller periodically. Contains information about the number of packets for which it received request for flow setup.	
RoleNotify	Notification for the role of controller at particular dpid. Master or slave etc. in the response of ofp_role_reply_handler	
EchoReply	Received in reply of global controllers periodic and regular echoRequest packet	
Error	Error code and message in case of error	
HostConnected	A host is added, added into dpid_host mapping as well as controller_host mapping	
DatapathLeave	Switch leave event, topology changed accordingly	
HostLeave	Host leave event, topology changed accordingly	
GraphInfo	Contains information about potential routes to other local controller domain, After confirmation from both controller added into topology. This is published in the beginning and later as and when there is any change in local controllers topology that may change inter domain route	

Table 9.2 Different messages sent from global to local controller

Message Type	Information	Reply message
HelloDown	Id of local controller assigned by global controller, needed for further communication with global controller	
RoleAssign	What is the role of the controller at that switch	
FlowMod	Information to set up path for inter domain routing.	
EchoRequest	Request from global controller to check if local controller is alive or not	EchoReply

Table 9.3 Ryu APIs to get topology information

API	Use
get_switch(dpid)	Returns information about that dpid
get_all_switch()	Returns information about all connected switches to that controller
get_link(dpid)	Return all links for that dpid
get_all_link()	Returns all links
get_host(dpid)	Returns host list connected to the dpid
get_all_host(dpid)	Returns all host connected to controller

Table 9.4 Open flow Events handled by local controller

Message Type	Information	Reply message
ofp_switch_features_handler	Contains the information that switch has been added. Informs global controller about it and sets flow entry in that switch, so that from next time onwards it could directly send packet to controller for flow setup.	DPConnected - to global controller add_flow - to switch
ofp_role_reply_handler	Controller send ROLE_REQUEST message to switch for the change in the role, if accepted ROLE_REPLY is returned, Local controller in turn sends this notification to Global controller	RoleNotify
ofp_packet_in_handler	Different type of information is received through this like host added, host left, route setup request for flow.If route is not setup switch forwards this packet to controller for decision making. If the destination is within local controller domain, Global controller is not consulted but if not, sends request for path setup.	HostConnected,HostLeave,PacketIn – To GC ; OFPPacketOut – To switch

Table 9.5 Ryu events handled by local controller

Message Type	Information	Reply message
EventSwitchEnter	Switch added event. Local controller asks ryu for topology data and creates a graph for routing. Also checks if there is any data for inter domain routing if yes send to G_C	GraphInfo – To G_C
EventSwitchLeave	Notifies local controller about switch leave, which removes this node from topology and informs global controller.	DatapathLeave – To G_C

References

[1] S. A. Diego Kreutz, Paulo Esteves Verssimo, "Software-defined networking: A comprehensive survey," 2015.

[2] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10. USENIX Association, 2010, pp. 3–3.

[3] N. G. J. S. L. P. M. Z. R. R. Y. I. H. I. T. H. S. S. Teemu Koponen, Martin Casado, "Onix: A distributed control platform for large-scale production networks."

[4] S. M. J. O. L. P. A. S. S. V. J. W. J. Z. M. Z. J. Z. U. H. S. S. Sushant Jain, Alok Kumar and A. Vahdat, "B4: Experience with a globally-deployed software defined wan."

[5] Y. G. Soheil Hassas Yeganeh, "Kandoo: A framework for efficient and scalable offloading of control applications."

[6] J. S. J. S. T. . K. Myungchul Kwak, Hyunwoo Lee, "Raon: A recursive abstraction of sdn control-plane for large-scale production networks."

[7] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, and B. Hao, "Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks," in *2014 IEEE 22nd International Conference on Network Protocols*, 2014.

[8] "Open flow," https://www.opennetworking.org/.

[9] "Mininet," http://mininet.org/.

[10] "Ryu sdn," https://osrg.github.io/ryu/.

[11] "A hierarchical control plane for sdn (software-defined networking) with a central controller and several local controllers." https://github.com/ppgirl/PARC.

[12] W. L. T. J. I. Farzaneh Pakzad, Marius Portmann, "Efficient topology discovery in software defined networks."

[13] "Iperf," https://iperf.fr/.

[14] "Matern," https://en.wikipedia.org/wiki/Mat%C3%A9rn_covariance_function.

[15] "Traffic flows," https://github.com/MaxiNet/trafficGen.

[16] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, 2013, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491193

[17] "Bayesian optimization," https://scikit-optimize.github.io/notebooks/bayesian-optimization.html.

[18] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *CoRR*, vol. abs/1012.2599, 2010. [Online]. Available: http://arxiv.org/abs/1012.2599