Scaling Software Defined Network Controllers Using Slicing Technology

Kanzhe Liu,
University of Waterloo

Abstract

Controller scalability of Software Defined Networks (SDN) is one of the fundamental research topics in SDN. In the industry, scalability could become an issue that restricts network performance, scale, and user experience. There are several conventional controller salability mechanisms, such as centralized design, distributed design, hierarchical design, and hybrid design [2]. Each of these design strategies has some pros and cons. In this paper, we will begin with a literature review of existing SDN controller salability schemes, and analysis of their features. Then, we will propose our new mechanism of scaling controller capability by using the network slicing technique. We will deploy the idea as proof of concept. The last, we will review those issues we meet in the current project, and make a plan about future improvement.

Introduction

Software Defined Networking (SDN) is a fast-growing technology. Implementations of SDN, such as data center networking, or LAN of big companies are currently online, and yield a considerable performance improvement. Unfortunately, due to its centralized network management feature, the SDN controller is always the computational hotspot of the entire network. In terms of acquiring better SDN controller scalability, researchers aimed at mechanisms that reduce or distribute the workload of a single controller, for instance, hierarchical distribution structures such as Kandoo, or allocating controllers for designated areas, like Onix, DCP, etc. The problem with these techniques is that data traffic may concentrate on physical spot at one period of time, and the advanced controller scaling mechanism will not be able to deal with this situation efficiently. With this in mind, we will propose and construct a new type of SDN controller scaling method based on network slicing techniques, in particular, FlowVisor. This paper has three sessions. First, we will investigate issues of controller scalability, and conventional solutions to those issues. Second, we will propose and rationalize our scaling solution, discuss features of techniques that we are going to use, and their potential. Finally, we are going to implement this new mechanism based on our research, and explore some improvements that can be made in the future.

General Review of Controller Scalability

The most significant trait of SDN is the separation of the data plane and control plane. This feature provides the ability to centralize network management and package routing functionality. The controller is the result of this design. It plays the role of brain and heart of an SDN implementation. By design, the most critical job of the controller is to receive requests from switches in its domain route and install the flow path with respect to an individual request. The problem here is that this design concentrates the computing power that routing function requires, and creates a computational hotspot. A conventional SDN controller architecture typically faces three challenges [1]. The first is the latency of switch-controller communications. This latency issue affects the flow rule installation time. Since the T-cam memory is relatively expensive and has limited resources, alternating and replacing leases in the routing table of a switch is an inevitable and constant behavior. This defect will hurt the networking throughput in the long run. The second issue is to choose the communication method in between controllers, in other words, the east and westbound API. This problem leads to traffic efficiency across the geographic location or networking scalability. The third issue involves the back-end database of the controller, it raises up the question of controller resource efficiency. These three problems are highly correlated. Thus, SDN scalability problem requires a systematic solution. Due to various networking conditions and chosen of SDN implementations, there is no standard or ultimate optimized scheme to solve this issue.

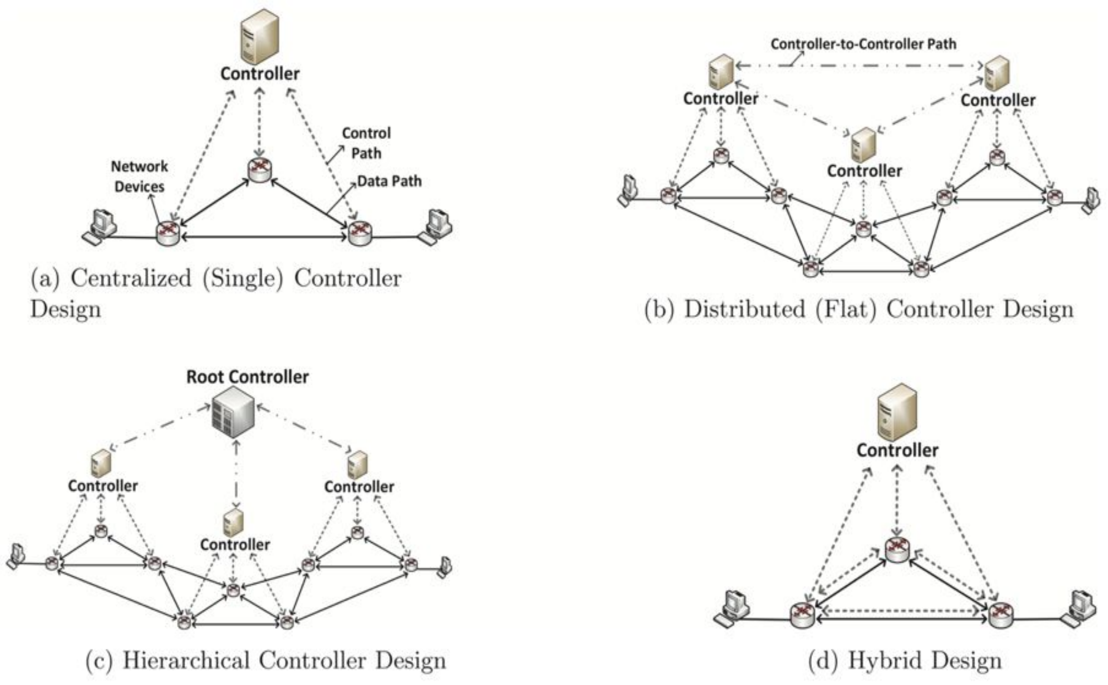


Figure.1 SDN Controller Topologies [2]

Karakus 2017 [2] categorized SDN scalability techniques into two major classes, mechanism related approaches, and topologies associated approaches. Mechanism related methods aim at boosting the performance of a single controller entity. Parallelism based optimization is one representative scheme, and its primary goal is to assign unnecessary work to different processes to reduce queuing [2]. Control plane routing scheme-based optimization is another way of doing this. This method tries to minimize controller routing events in order to boost scalability of SDN networking.

Current research shows that improving computational resources on a single controller is optimal but has a limited effect on scalability performance [3]. Therefore, optimizing controller scalability by using topology related approaches is necessary. This is true even if we consider that controller computing resources are unlimited.

There are several topological techniques that compliment controller design such as centralized, hierarchical distributed, flat distributed, and hybrid [2]. We will discuss characteristics of each topology design in next following chapters, and then we will rationalize the motivation of our new proposal.

Centralized Design

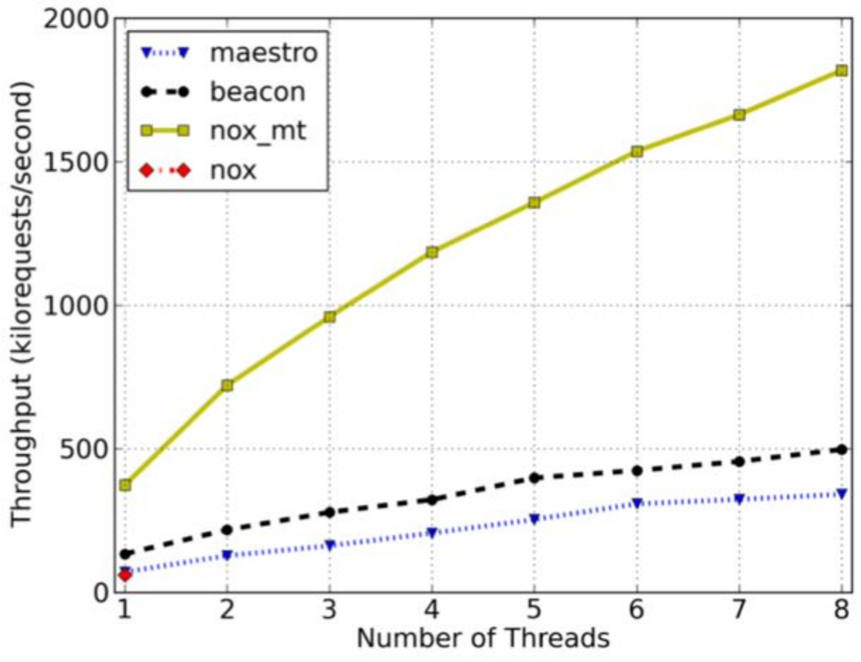


Figure.2 Performance throughput comparison of NOX_MT and other controllers [8]

The centralized design is the simplest and the most conventional scheme. Its design principles follow the mechanisms related approaches and its features turn out to be one of the ideal solutions for small to medium size LAN. The representative of this controller design is Nox. From a general perspective, Nox is a generic OpenFlow based controller, and it heavily relies on controller-to-switch communication for network routing. In order to maintain the network consistency, the view of network state remains global [5]. Research [7] indicated that a single Nox controller can only handle 30k requests per second. Regarding this problem, researchers came up with two solutions. First, use a group of Nox controller instances to deal with a large load of requests. In [7], researchers prepared a testing environment for two million virtual machines. In order to deal with predicted requests from those VMs, there were 667 controller instances to handle network flows. Second, upgrade to NOX-MT (multi-thread) regarding boosting performance [8]. It employs some basic parallel techniques such as multi-threading or I/O bashing, regarding improving I/O performance [2, 8]. The benchmark result shows drastic improvement. The benchmark was almost four times better than original NOX controller throughput [8]. Further research also indicates that a single node controller architecture still faces the bottleneck of performance, when it deals with a large scale network [3, 8].

Distributed Flat Design

Distributed flat design contains a set of parallel controllers, each one of them charges a sub-domain of the network. Any arbitrary controller has the global information of the entire network, and it treats adjacent sub-networks as logical entities. A typical distributed flat control plane is Hyperflow controller. This controller is a logically centralized but physically spread [9]. As we defined before, each Hyperflow must maintain the state of the global network regarding two reasons. The first reason involves controller scalability. The second reason is computing power reinforcement. If one controller is down, adjacent controllers will divide that particular region of network nodes, and take over the work of the failed controller.

This design drastically extends the scale of an SDN controller and brings robustness to the scheme. However, it also brings up some challenges. The most significant defect is that this mechanism requires a highly efficient algorithm to solve a superlinear computational complexity increment that grows with networking size [2].

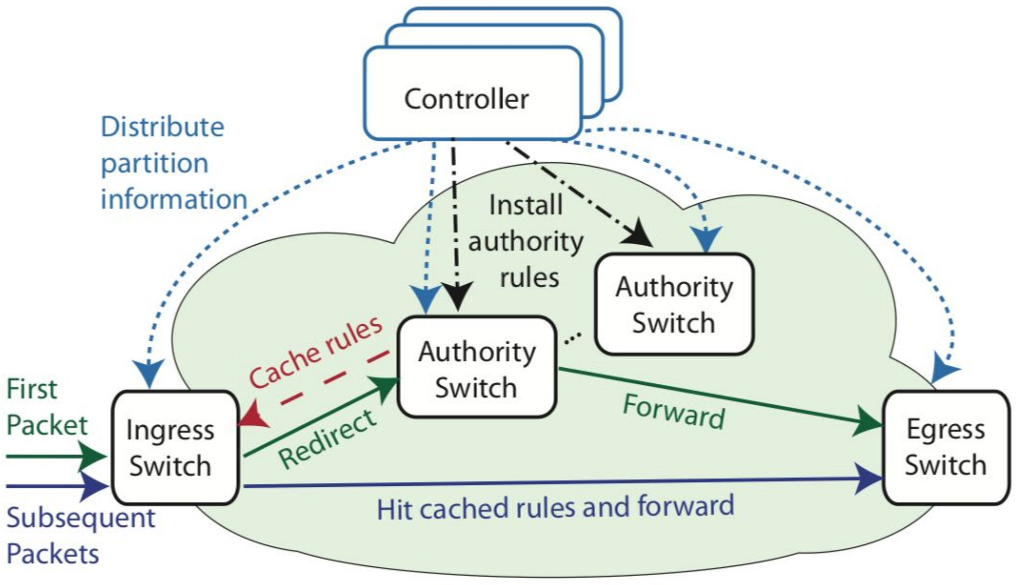
Hierarchical Design

Hierarchical controller design works similar to flat controller design, but instead of a controller-to-controller communication channel, the hierarchical architecture uses a "root" controller on top of local controllers. Under this circumstance, local controllers do not maintain a global view, and the centralized higher tire controller (root controller) manages and dispatches packages across sub-networks. A typical hierarchical designed controller is Kandoo. Kandoo has the controller topology that divides the whole network into many smaller chunks. However, unlike flat topology, the design purpose of Kandoo's local controller is to manage frequent events in a restricted domain, thus isolating them from the root controller. Its root controller deals with the traffic flow that across global networks [10].

However, this hierarchical controller topology may have difficulty using shortest path between nodes, and this problem also is known as "path stretch problem" [2, 4, 5]. It also cannot help for control application which works on global network-wide [10].

Hybrid Design

Hybrid controller design often involves both the control plane and data plane, and this also is the reason of its name "hybrid". This design has two distinguishing features. First, the control plane of this architecture plots and distributes rules to authority switches. Second, the data plane charges for packets dispatching according to the previously authorized rules from control plane [12]. In other words, a hybrid controller like DIFANE delegates some functionality such as exchanging flow rules, managing flow tables, etc. to the data plane, thus reducing the workload of the controller, and boosting up the controller scalability [2]. Due to the extra computation on the data plane, this design requires some additional resources on



data plane [12].

Figure.3 DIFANE flow managment diagram [12]

Motivation of the New Proposal

Through demonstrations of mainstream topology-based controller scalability architectures, we can clearly see two facts about the status quo of SDN industry. First, each one of the controller scaling schemes has limitations. Their constraints shape the traits and characteristics of each controller, and restrict the usable scenarios for these controllers. Second, we can tell that each one of them are restricted to functioning within a certain framework. Each technology has a complete set of services and functionalities, thus, it is impossible to combine those technologies without source code modification and tuning. The current circumstance inspires the question, can we make a framework that be able to gather the features of other frameworks while avoiding defects? Regarding this problem, we propose a new controller scaling architecture that has potential to achieve this goal. The next section is about the blueprint of our new mechanism.

Proposal of Controller Scalability by Using Network Slicing

We attempt to discover a new type of controller scalability mechanism regarding avoiding issues such as superlinear complexity growth, or path stretch problem. Also, we would like to make this new scheme has implementation flexibility feature, and open the door for merging with other technologies in the future. This session will introduce this new scheme using modules in the following order: introduction, related work, demonstration of networking topology, business process, design feature, dilemmas of scale and efficiency, and finally we will face some challenges of implementation.

Toward to the Third Dimension

SDN controller scalability by slicing (SCSS), as its name suggests, is a mechanism that splits SDN controller workload into different divisions. From topology perspective, previous approaches we introduced above scale the workload in “horizontal” way (flat design), or “vertical” way (hierarchical design). In this design, we will engage in the “third” dimension of scalability which parallels not just the controller, also could be the entire network. The key tool of this design is network slicing, such as FlowVisor. The conventional usage of network slicing is to separate the entire network into several logical areas. In SCSS design, however, we propose to use slicing technique to collect a set of busy switches that constantly updates its flow table. Furthermore, we can make an arbitrary number of "copies" of the whole network. Each copy has a controller that manages a subset of total traffic flows. On top of the control plane, there is a central dispatch controller, or central controller program that stores and maintains states of the global network and slicing management. By combining all these features, we can add a new dimension of controller scalability, and remain quite flexibility. We conducted a test concerning proving this concept. The test-bed is Mininet. We setup a linear network with eight switches and 80 hosts (10 hosts on each switch), and those hosts have an index from 0 to 79. We split the network into two even halves by FlowVisor, and all flow-space matching rule is set to "any", which means the matching condition is the wildcard. In the control group, both divisions shared the same piece of controller resources, and two divisions owned their exclusive controller in the experimental group. During the test, a host pinged selected higher index of other hosts until the end of the host list. There are two purposes of this test. The first goal is to see if wildcard flow-space rule is working correctly. The second goal is to compare how fast these two controllers handle 1600 requests. We did three rounds of the trial. We observed that the difference between the two groups was not significant. The mean of experimental group finishing time was 131.83 seconds, and the mean of the control group was 134.816 seconds. When you consider the rudimentary test environment where every virtual machine was using the same hardware resources, this mechanism could have potential on a real cloud-based system with sufficient computing power.

From a particular switch perspective, when a packet arrives, the highest priority flow-space and its connected controller is the best option for this router to query. This proposal manipulates this property of flow-space. Like what we did in the last experiment, we defined all flow-space matching conditions as the wildcard, which means this flow-space can forward packets to any arbitrary port in the switch, and switches own multiple identical flow-space rules but a different priority. We purposely skip around the isolation feature of the network slicing tool, regarding a smooth transition of packets.

Related Work

The network slicing technique is a conventional methodology of network management. Its main functionality is to divide and isolate network into different pieces. This feature can achieve many goals. In the homepage of FlowVisor project on Github, managing network bandwidth by using network slicing is a demo project of FlowVisor [12]. However, this example created network slices and flow-space by manipulating pre-acquired network states. Thus, it is not a generic networking management application.

There are several research papers that aim at managing traits of FlowVisor and network virtualization. Guo 2010 [14] is the one that tried to implement FlowVisor for the virtualized data center (VDC). In this paper, researchers proposed to provide VDC services to tenants by using FlowVisor, to achieve bandwidth guarantee, and prevent tenants cross-talking.

The principle of our new proposal is quite similar to the idea of the demonstration. However, instead of focusing on tenancy isolation and bandwidth guarantee, we are going to manipulate the feature that each slice has a designated controller for scaling purpose. The most significant improvement from the previous work is to setup a scheme that continuously checks the network states and power consumption of controller. It will then dynamically turn slices on and off to scale network controller accordingly.

Basic Networking Topology

Figure .4 shows the basic structure of the SCSS. The central controller program monitors the status of the entire network and rises new slice according to switches' state. As we can see here, there are several slices in the structure. Each slice has a group of flow-space which response to a set of the hotspot, and an assigned controller for handling packets in this domain. The central controller program sits in the middle and does its job. By putting all slices together, we can get a complete picture of the whole network.

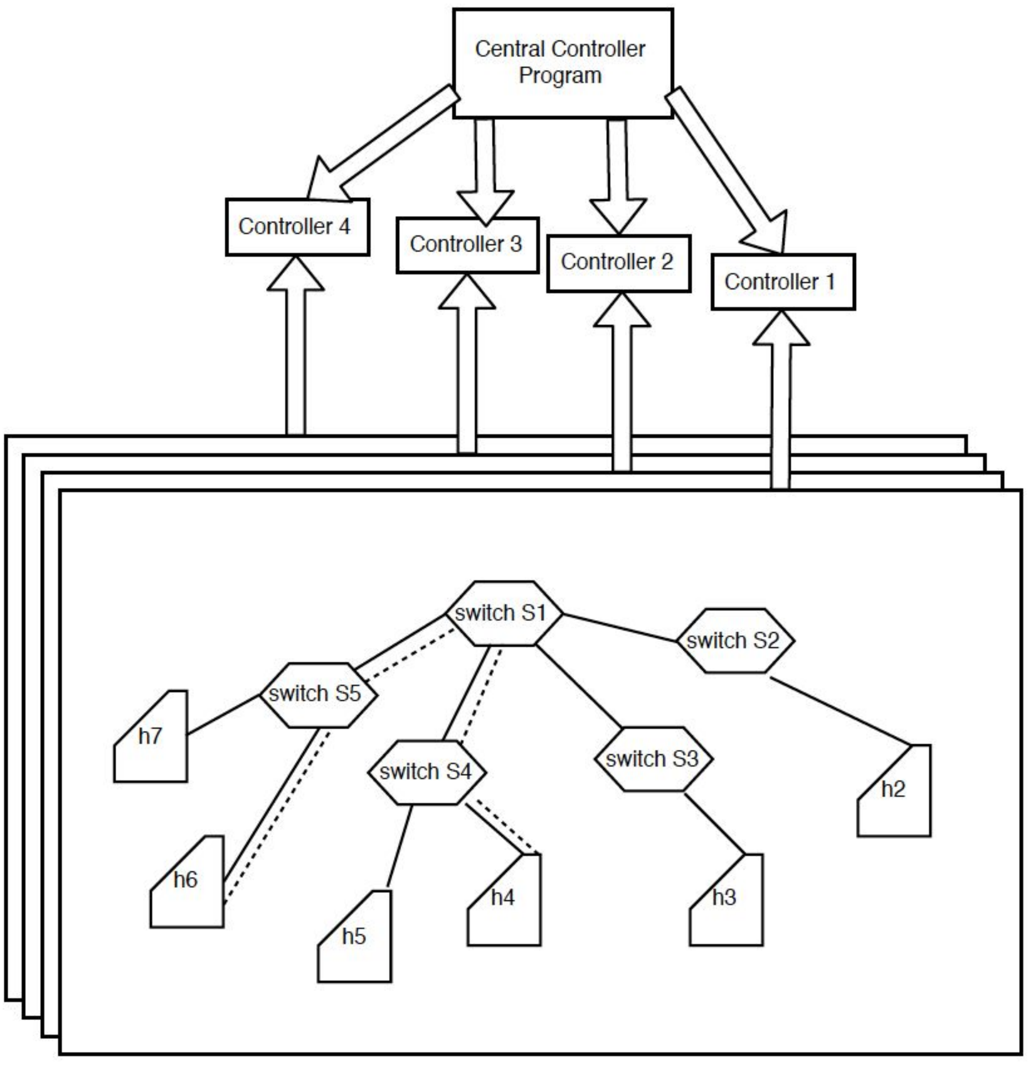


Figure.4 Structure diagram of SCSS. The dashed line represents the flow-space on this slice with higher priority

Business Process of Central Controller Program

The most critical part of this project is the central controller program. There are four modules in the application, network state acquiring module, slice-state database, slice-state maintenance, and slice operator. When the primary controller is up and running, it periodically goes through a set of business process. The first mission is to check the status of each slice. The program gets slice information and switches information accordingly, then decide to keep the status quo or delete the slice. In this prototype, we set that the deletion will occur if the ratio of "busy switches" on a particular slice is lower than 50%. If a slice triggers deletion condition, the central controller will remove the slice from the network, terminates the controller process, and eliminates the slice record from the database.

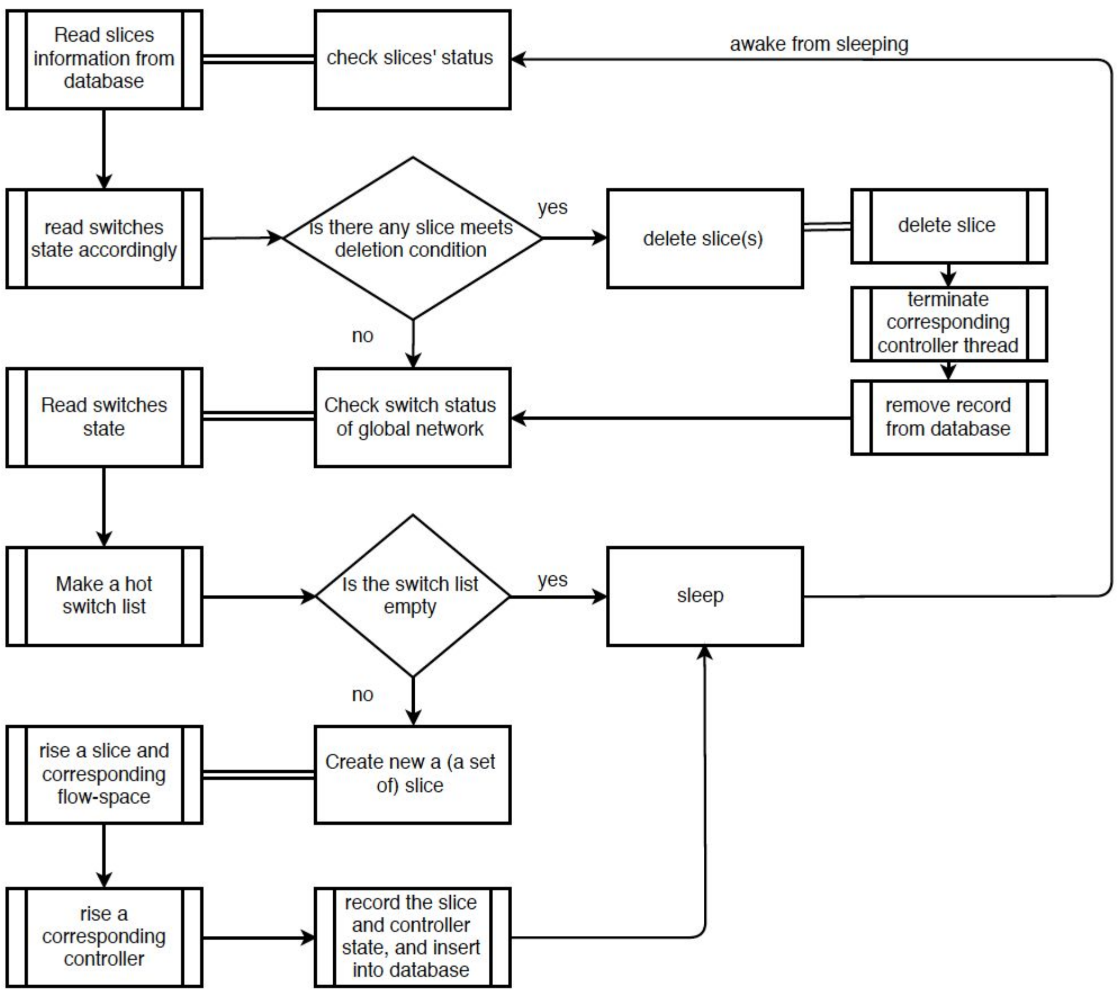

Read slices information from database
check slices' status
awake from sleeping
read switches state accordingly
Is there any slice meets deletion condition
yes
delete slice(s)
delete slice
terminate corresponding controller thread
no
Read switches state
Check switch status of global network
remove record from database
Make a hot switch list
Is the switch list empty
yes
sleep
no
rise a slice and corresponding flow-space
Create new a (a set of) slice
rise a corresponding controller
record the slice and controller state, and insert into database

Figure.5 Business process of central controller program. The double line relates the event and the detailed process of the event

Second, the central controller makes a “hotspot” list via network status and sends it to slice operator module for further process. In this version of the prototype, we defined a hotspot as a switch with a certain number of flow-rules installed, and 50% of flow-rules have a lifespan less than 30 seconds. Third, the slice operator filters out controllers that already in a slice, and ready to make a new slice for those “orphan hotspots”.

At last, the central controller adds the new slice and its corresponding controller into the slice database, and get into sleep mode. It wakes up once in a while to save some power consumption.

Design Features

Based on the explanation above, we can easily find that “flexibility” is the most distinguished feature of this new controller scaling mechanism. There are two ways to explain this trait. First and foremost, the controller scaling function is totally decoupled with its primary structure. This characteristic lets controller choosing become very flexible, and it makes technology combination possible. The slicing layer is a network virtualization layer which built on the OpenFlow protocol. It sits between hardware and software resources. From a controller point of view, the FlowVisor is merely an entrance to access information [13]. Implementing a controller is also an easy task on FlowVisor. Consider the scenario that the slicing scaling technique combines with a powerful controller structure, such as NOX-MT. It would be a high-performance boost. This characteristic comes with FlowVisor technology, and it makes a great sense to the project.

The second important feature is the dynamic adaptation of network and ultra large computational power. This trait is one of the design purposes as well. The central controller program slices network according to hardware status. There is no limit on slicing rule by default, and in some extreme case scenario, the central controller can create a massive computing power delivery, such as one controller deals with one switch. This feature only works on paper by far, but still, it shows the mechanism has no limit if computing power allows.

Even if we consider the brute force computing power delivery as one of the features of the system, the algorithm efficiency is still a big concern of the design. The next chapter will attempt to explore this issue.

Dilemmas of Scale and Efficiency

The basic structure a business process provides a rough shape of the proposal. However, we still have a lot of detailed problems. First, should each slice contain global network information or just a state of local switches? This question is highly coupled with routing algorithms. In the current prototype, we used a very fundamental OVS-controller. This controller does not have very a advanced algorithm, so in this round of implementation, a slice only has hot spots as its local-switch members. If a packet's destination is out of the boundary of the current slice, it will rely on the result flooding algorithm. By contrast, to avoid the problem of overly stretched paths, it is necessary to keep a copy of global network state in each controller. This scenario is quite close to the issue we have in the distributed flat design, but the situation here could be worse. Because we have no controller-to-controller communication in the blueprint, there is no way for a sub-controller to send the synchronization message to an adjacent spot, and the next sub-controller has to recompute the entire route for the same packet. This is an urgent issue we need to solve when introducing complex routing algorithm. The second is also the extension of the first problem, which is how much global state should a controller preserve. This problem leads to a tradeoff between computing time and all-pairs shortest-paths in the network. This issue is essential to routing efficiency of a single sub-controller, and it takes place in general network performance.

Despite these known problems in our design, a prototype is required for proof of concept. We choose to use some fundamental tools and framework to build the model for further research. The next chapter is the detailed information about it.

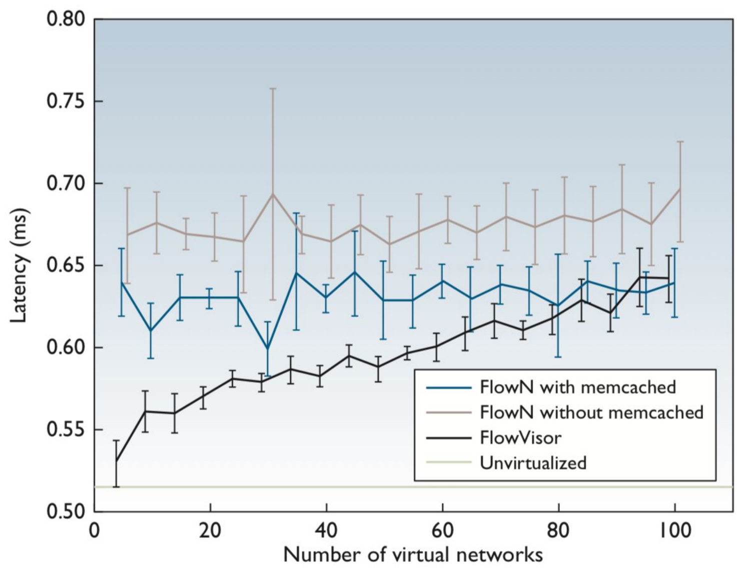
Implementation and Challenges

The SCSS provides a new opportunity to manage the growing consumption of controller computation, and we attempted to prototype it by using existing technologies. We chose to use FlowVisor and OVS-Controller on Mininet. These are commonly used tools. We realized the significant functionality of the prototype, but we also discovered some fatal flaws of currently using frameworks. First and foremost, due to the pool test-bed, Mininet, the communication between hosts will be completely shut down when the new slice and corresponding controller boost up. The only solution for this issue is to restart the Mininet. Thus, under current circumstance, it is impossible to conduct any meaningful experiment for this dynamic slicing tool. Second, since we scale controller by using the slicing framework, it is tough to use the same tool to divide network again. The potential solution is to integrate slicing functionality with scaling functionality, but it demands on extra work for making an API which respects to this issue. Third, one of the purpose controller scalability is to solve the problem of controller computational hotspot. In our prototype, the controller merely runs on a thread, and the whole network nodes share the same piece of hardware. Thus, we entirely missed this part of the content. Respect to these issues we have, it is necessary to list what we should do in the future.

Future Work

During this project, we proved the concept of scaling controller by using network slicing technique. We also implemented the prototype and showed its potential of being commercial use. However, there are also some defects that caused by the unrefined environment. Thus, we have several items on the to-do-list, regarding further research.

First and foremost, this project needs to migrate to a proper cloud-based platform, such as SAVI. In the new test-bed, we will have real switches and adequate computing power. Therefore, it is possible to solve broken connection problem due to the power consumption of controller VM. Also, with proper cloud server, we will be able to bring back the controller performance monitoring function, and complete it design purpose. Furthermore, the performance test will be available, and we can see how much improvement



it can achieve.

Figure.6 Latency comparison of FlowN and FlowVisor [15]

Secondly, a comparison test of FlowVisor and FlowN is necessary. Drutskoy 2013 [15] expressed that FlowVisor does have shorter latency time in a small size network. However, in the FlowN scales better when the network reaches a certain number of slices. Thus, it is important to know how they will perform in our new environment.

Third, due to the design purpose and feature, the virtual network environment will be very dynamic and unstable. Thus, implementing the controller-to-controller communication is not a very feasible choice. Like we stated in dilemmas of scale and efficiency, a proper routing algorithm is required, regarding diminishing duplicated routing computing and boosting productivity to the new architecture.

Forth, we need to refine and further develop this project regarding the network virtualization function. Network virtualization function is a very powerful and widely used scheme. However, in our project, the controller scalability takes the spot of that. Thus, it is necessary to create an API for combing actual slicing technique with our architecture.

The last but not least, experiment and prototype by far are running on a small scale of network emulator. However, the network scale could be varying in reality. It is essential to know the price ratio of controller performance and the cost of maintaining global network state.

Conclusion

Through this project, we demonstrated the potential of network slicing technique respect to controller scalability. Due to the possible way of preserving global network state, this mechanism provides opportunity of diminishing path stretch problem. However, we need further experiment under appropriate environment to test the performance and its limit.

Reference

[1] Sezer, Sakir, et al. "Are we ready for SDN? Implementation challenges for software-defined networks." *IEEE Communications Magazine* 51.7 (2013): 36-43.

[2] Karakus, Murat, and Arjan Durresi. "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)." *Computer Networks* 112 (2017): 279-293.

[3] Shah, Syed Abdullah, et al. "An architectural evaluation of SDN controllers." *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 2013.

[4] Cowen, Lenore J. "Compact routing with minimum stretch." *Journal of Algorithms* 38.1 (2001): 170-183.

[5] Fu, Yonghong, et al. "A hybrid hierarchical control plane for flow-based large-scale software-defined networks." *IEEE Transactions on Network and Service Management* 12.2 (2015): 117-131.

[6] Gude, Natasha, et al. "NOX: towards an operating system for networks." *ACM SIGCOMM Computer Communication Review*38.3 (2008): 105-110.

[7] Tavakoli, Arsalan, et al. "Applying NOX to the Datacenter." *HotNets*. 2009.

[8] Tootoonchian, Amin, et al. "On Controller Performance in Software-Defined Networks." *Hot-ICE* 12 (2012): 1-6.

[9] Tootoonchian, Amin, and Yashar Ganjali. "Hyperflow: A distributed control plane for openflow." *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. 2010.

[10] Hassas Yeganeh, Soheil, and Yashar Ganjali. "Kandoo: a framework for efficient and scalable offloading of control applications." *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012.

[11] Yu, Minlan, et al. "Scalable flow-based networking with DIFANE." *ACM SIGCOMM Computer Communication Review*40.4 (2010): 351-362.

[12] https://github.com/onstutorial/onstutorial/wiki/Flowvisor-Exercise

[13] Sherwood, Rob, et al. "Flowvisor: A network virtualization layer." *OpenFlow Switch Consortium, Tech. Rep* 1 (2009): 132.

[14] Guo, Chuanxiong, et al. "Secondnet: a data center network virtualization architecture with bandwidth guarantees." *Proceedings of the 6th International COnference*. ACM, 2010.

[15] Drutskoy, Dmitry, Eric Keller, and Jennifer Rexford. "Scalable network virtualization in software-defined networks." *IEEE Internet Computing* 17.2 (2013): 20-27.