Wireless Project Report

Scalability of Software Defined Network

Lin Zhou
5100309339

Lei Zhang
5100309405

2013/06/01

Abstract

Software Defined Network decouples control plane from data plane and facilitate the control to the whole network.With the number of switches in SDN increases,there's no way to avoid the question of scalability.We focused on this problem and analyzed several solutions to this problem and finally formed our model HMKH(Hybrid Model Based on Kandoo and Hyperflow).HMKH can change dynamically with the network requests and solve the scalability problem in many cases.

1 Scalability in SDN and Existed Solutions

1.1 Brief Introduction to SDN

SDN is an approach to building computer networking equipment and software that separates and abstracts elements of these systems. These elements are called the control plane and the data plane. SDN allows network administrators to manage network services more easily through abstraction of lower level functionality into virtual services. This replaces having to manually configure hardware. This has become more important with the emergence of virtualization which an enterprise data center may need to create and configure virtual machines (VMs) remotely, and configure firewall rules or network addresses in response. Many approaches exist to resolve this issue such as Virtual LANs but this may also introduce management issues.SDN allows network administrators to have programmable central control of network traffic without requiring physical access to the network's hardware devices. SDN decouples the system that makes decisions about where traffic is sent (the control plane) from the underlying system that forwards traffic to the selected destination (the data plane).The inventors and vendors of these systems claim that this technology simplifies networking and enables new applications, such as network virtualization in which the control plane is separated from the data plane and implemented in a software application.

1.2 The Origin of the Scalability Problems in SDN

There have always been concerns about performance and scalability since its inception.
The common perception that control in SDN is centralized leads to concerns about SDN scalability and resiliency. After all, regardless of the controller capability, a central controller does not scale as the network grows (increase the number of switches, flows, bandwidth, etc.) and will fail to handle all the incoming requests while providing the same service guarantees. Additionally, early benchmarks on NOX (the first SDN controller), which showed it could only handle 30,000 flow initiations per second [1] while maintaining a sub-10 ms flow install time, intensified such concerns.

1.3 Current Solutions to this Problem

1.3.1 DIFANE

The DIFANE architecture consists of a controller that generates the rules and allocates them to the authority switches. The basic architecture of DIFANE is as shown in Figure 1. Upon receiving traffic that does not match the cached rules, the ingress switch encapsulates and redirects the packet to the appropriate authority switch based on the partition information. The authority switch handles the packet in the data plane and sends feedback to the ingress switch to cache the relevant rule(s) locally. Subsequent packets matching the cached rules can be encapsulated and forwarded directly to the egress switch.

Advantages:
(1) DIFANE achieves small delay for the first packet of a flow by always keeping packets in the fast path.
(2) DIFANE achieves significantly higher throughput than NOX.
(3) DIFANE scales with the number of authority switches.
(4) DIFANE recovers quickly from authority switch failure.

Disadvantages:
(1)A number of authority switches are needed for the large networks we evaluated.
(2)DIFANE does not address the issue of global visibility of flow states and statistics. The types of management solutions we would like to enable rely on global visibility and therefore it is unlikely they can be built on top of DIFANE.

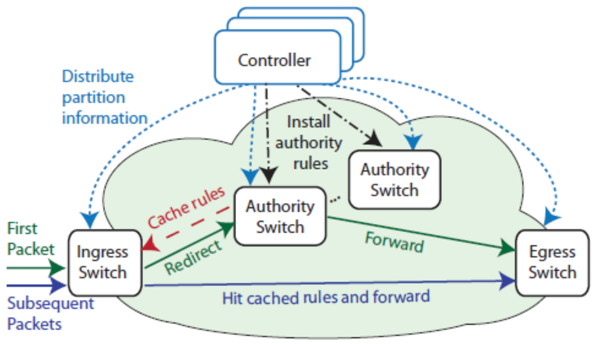

Figure 1: DIFANE flow management architecture

1.3.2 DevoFlow

DevoFlow enables scalable implementation of these solutions by reducing the number of flows that interact with the control-plane. By pushing responsibility over most flows to switches and adding efficient statistics collection mechanisms to identify significant flows, which are the only flows managed by the central controller, it can solve the scalability problem.

Advantages:
It can reduce the load of the controller.

Disadvantages:
(1) The significant flows should represent a small fraction of the total flows, but how many flows would be sufficient to achieve the desired results in different environments.
(2)It is hard to build a efficient statistics collection mechanisms.

1.3.3 HyperFlow

HyperFlow is logically centralized but physically distributed,as shown in Figure 2.
HyperFlow provides scalability while keeping network control logically centralized: all the controllers share the same consistent network-wide view and locally serve requests without actively contacting any remote node, thus minimizing the flow setup times. Additionally, HyperFlow does not require any changes to the OpenFlow standard and only needs minor modifications to existing control applications.

Advantages:
(1)HyperFlow enables network operators deploy any number of controllers to tune the performance of the control plane based on their needs.
(2)HyperFlow keeps the network control logic centralized and localizes all decisions to each controller to minimize control plane response time.

Disadvantages:
(1)HyperFlow doesnt change the number of the switches which controller controls.

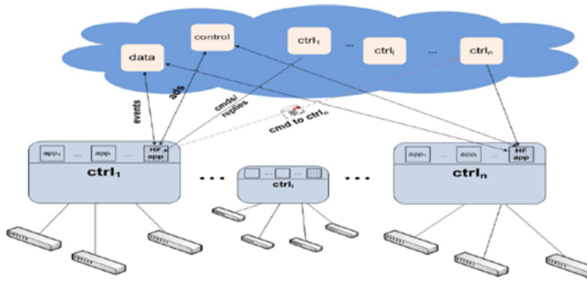

Figure 2: High-level overview of HyperFlow

1.3.4 Kandoo

Kandoo is a framework for preserving scalability without changing switches.

Kandoo creates a two-level hierarchy for controllers: (i) local controllers execute local applications as close as possible to switches, and (ii) a logically centralized root controller runs non-local control applications. As illustrated in Figure 3, several local controllers are deployed throughout the network; each of these controllers controls one or a handful of switches. The root controller, on the other hand, controls all local controllers.

Advantages:

The main advantage of Kandoo is that it gives network operators the freedom to configure the deployment model of control plane functionalities based on the characteristics of control applications. Briefly to say, Kandoo makes the offloading of control applications efficient and scalable.

Disadvantages:

The main disadvantage of Kandoo is that it cannot help any control applications that require network-wide state (even though it does not hurt them, either).

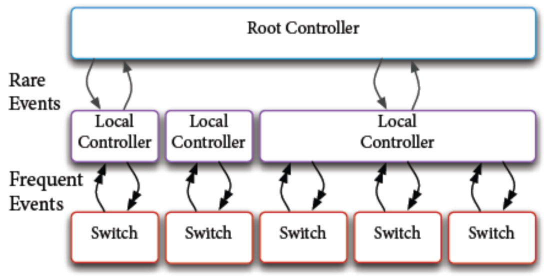

Figure 3: Kandoo's Two Levels of Controllers. Local controllers handle frequent events, while a logically centralized root controller handles rare events.

2 Goals and Model Design

As talked above,current solutions to scalability have their own advantages and disadvantages.The purely centralized Devoflow and Difane can only increase the scalability to some extent and switches are modified;The transition from centralized to distributed mechanism Kandoo can solve the problem where the percentage of requests from switches for the network-wide view information is relatively low;The totally distributed Hyperflow can scale with the number of switches theoretically but suffer from another problem of synchronization time which would affect the scalability capability a lot.

To solve this problem,we should have a model which can scale regardless of the percentage of requests for network-wide view.First,this model should scale to infinity if the number of switches goes to infinity while still offer the effective control of the network.This situation is met in Data Center where the density of switches is pretty high but the area is relatively small.Second,this model should be robust to switch failure and controller failure and has mechanism to solve these failures.Third,this model should be robust to network partition and fusion.

To achieve the above goals,we devise the model in figure 4.Simply speaking,we replace each controller in Hyperflow with the hierarchical model of Kandoo in a wireless instead of wired connection.Compared to Hyperflow,the number of switches each controller with network-wide view increase and compared with Kandoo,the scalability capability improves greatly.

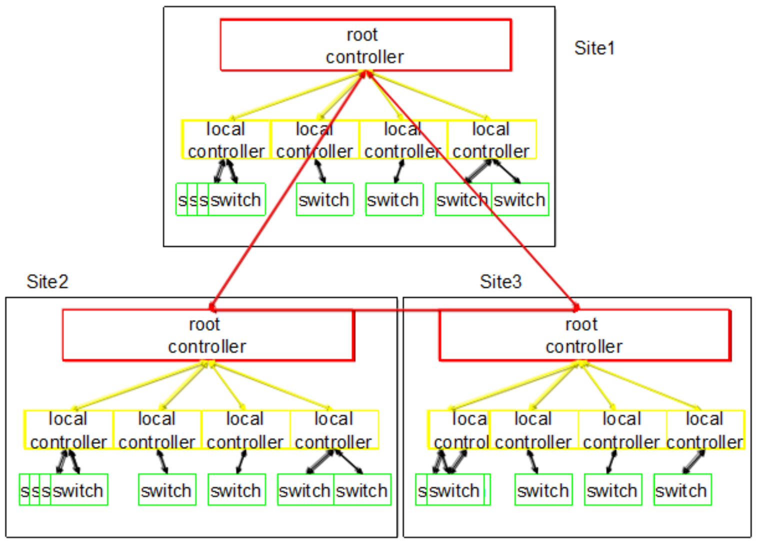

Figure 4: HMKH Model

3 Details of the HMKH

3.1 Assumptions

Before we further talk about the details of this model,we have several assumptions to make.

1. The communications are all wireless and wireless communication detail is not the coverage of this report.

2. Any switches controlled by a root controller in

a site is in the control range of that root controller.

3. The direct neighbours of any root controller are within the communication range of the root controller.

And these assumptions are the base for the discussions below.

3.2 Controllers and Applications

We have two kinds of controllers in our model:local controllers and root controllers .

- **local controller**
  Each local controller gathers information about the switches controlled by it including their neighbour and the distance between a switch and their neighbours,save these information and send it to its root controller.

- **root controller**
  Root controller in a site not only has the information of switches in its site and the topology of the connections(the direct neighbour of a switch and the distance) from its local controllers but also get information about other sites from communication with the other root controllers.Therefore,every root controller has the information of all the switches in the network.

And respectively,we sort information and applications into two kinds.

- **local-view information and local applications**
  local information:information kept by a local controller,namely the information about the subnetwork of the local controller.
  local applications:apps need local-view information

- **network-view information and global applications**
  network-view information:information kept by a root controller,namely the information about the whole network.
  global applications:apps need network-view information

4 Implementations of HMKH

Each switch is openflow switch and act as forwarding elements.We have initiation to set up the network,the procedure to keep the network and the mechanism to deal with failures in the network.

4.1 Initiation

Before initiation,all the switches and controllers are already where they are and have their own unique device number(used as ip).The initiation is divided into two steps which happen simultaneously:

1. root controller finds local controllers:
   Each root controller broadcasts a request included its device number and location information.Once a local controller receives this type of request for the first time,it sets the source root controller as its root controller, note down the root controller device number in its root controller register and calculate its distance to its root controller.

2. local controller finds switches:
   Each local controller broadcasts a request included its device number and local information.Once a switch receives this type of request for the first time,it sets the source local controller as its local controller and note down the local controller device number in its local controller register.

After this,we have the network established as figure4 on page3.

4.2 Holding Procedure

4.2.1 Periodic Communications and Failure-free Mechanism

Each root controller broadcast with period T_c with its device number and a sequence number to show its existence.Once a local controller receives this from its root controller,it responds with its own deice number and a sequence number.

- **Root Controller Failure**
  Once a local controller can't get this kind of information for 3 consecutive periods,it believes the root controller fails and broadcasts with the root controller device number,its own device number and its distance to the original root

controller to other local controllers in the same site showing that it wants to be the new root controller.
The local controller li changes its root controller to the source local controller only when it receives a request with smaller distance to the original root controller than kept candidate's distance to the original root controller and stops sending request to be a root if it sends before.
A candidate local controller becomes the root controller only when it gets response from all the other local controllers in the site which is indicated by that it never receives request to be a root from other local controllers in the same site.After this,the local controller which is closest to the original root controller will be the new root controller.

- **Local Controller Failure**
Once a root controller can't get response from a local controller for 3 consecutive periods,it believes the local controller fails.
The local controller assigns the switches originally controlled by the failed local controller to the local controller which is nearest to the failed one by sending command to the assigned local controller.And the local controller broadcast the local controller change information to the desired switches.Once a switch gets that,it changes its local controller device number to the source local controller.

4.2.2 Network Change Information

The main idea is that the root controllers all hold the network-wide information.Thus,once the network changes,whether the controllers or switches change,the root controllers should synchronize these information.To achieve this,once a root controller detects such change,it should broadcast a network change information to all the other root controllers.
If a switch is detected failed by a local controller,it reports this to its root controller.The root controller broadcast once it gets this report.This is also true for a switch is added.
Another important use of this mechanism is in network fusion.If we fuse network A with B,the root controllers will get information from the other network.Once it gets this,it broadcasts to other root controllers so that all the root controllers in the new network will have the new network-view information.

4.2.3 Applications

We use application route here to explain HMKH more specifically.As discussed above,we have local route and global route apps.
Local route application asks route within the range of the same local controller.Once a local controller gets this kind of request from its controlled switch,it calculates and responds with the path.Then the switch broadcasts the data along the path.
Similar things happen when global route application asks for route.But the difference is that local controller first forward this request to its root controller.The root controller then calculates the path and sends to the local controller.Finnaly,the local controller sends the request switch the full path.

4.2.4 Data Cache Mechanism

To avoid data forward failure during the period of controller failure ,we have the cache mechanisms.

- local controller failure:if a switch can't get response from local controller,it will cache the data and retransmit the request until the local controller is okay.
- root controller failure:once a local controller finds the root controller fails,it will cache all the requests to the root until a new root is elected or it becomes the root controller.

5 Evaluations of HMKH

We first give assumptions about the ability of controllers:each controller can process at most k requests per second.
To evaluate HMKH,we should first have a scenario.

- totally n switches to serve and each switch sends s number of requests per second
- the probability for each switch to ask for network-view state is p

In this scenario,we have $s * n$ requests to controllers and $s * n * p$ of them need network-view state.Therefore,we need $m = \frac{s*n*p}{k}$ root controllers for HMKH and we need $\frac{s*n}{m*k}$ local controllers.

Compared with Kandoo which only has one root controller,we have the advantage of scalability for if m is greater that 1,Kandoo will not work well.
Compared with Hyperflow,we need $\frac{s*n*(1-p)}{k}$ less root controllers which have high overhead to synchronize network-wide state.

6 Conclusions

HMKH has better performance over the current mechanisms like Kandoo and Hyperflow as shown in evaluations.The advantage of it is that it takes advantages of other mechanisms and avoids the disadvantages successfully. The question of scalability lies not only in how many switches a controller system can serve,but also the number of switches a controller with network-view state can serve.Hyperflow successfully achieves the former while Kandoo does better in latter.However,with HMKH,we do both.
But before the use of our model,the number of root controllers and local controllers should be calculated or estimated given the number of requests from switches and the percentage of requests for network-view state.With that information,we can better determine the number and location of controllers to better serve the switches and minimize the overhead,which is not included in the coverage of this report.
If in a rather dense network where all the root controllers can reach each other via wireless communication,HMKH can change from Kandoo to Hyperflow given the requests from the switches dynamically.
If the sum of requests to two root controllers is less that one threshold,we could incorporate these two root controllers:one of them with smaller number of requests first send incorporate request to the other root controller.The destination root controller once gets this will first check whether combination is reasonable by compared to the threshold.If reasonable,it sends a incorporation command to the source root controller.The source root controller once gets this will broadcast to its original local controllers of the root controller change information and then change itself as a local controller of the destination root controller.In this way,two root controllers incorporate.
For the opposite situation,if a root controller gets requests more than another threshold,it will broadcast to its local controllers to elect another root controllers in the same mechanism as root controller failure.After this,a root controller is separated into two.
With this two process,a network with very small percentage of requests to root controllers will finally become a Kandoo model.Similarly,a network with nearly 100 percentage of requests to root controllers will finally form a Hyperflow model.The dynamic change between this can make best use of each root controllers.

References

[1] A. Tavakkoli et al , "Applying NOX to the Datacenter," Proc. ACM HotNets-VIII Wksp, 2009.
[2]T.Koponen et al,"Onix:A Distributed Control Platform for Large-Scale Production Networks," Proc. 9th USENIX OSDI Conf, 2010, pp. 1C6.
[3]Minlan Yu et al,"Scalable Flow-Based Networking with DIFANE,"Proc. ACM SIGCOMM Computer Communication Review,2011.
[4]Tootoonchian et al,"HyperFlow: A distributed control plane for OpenFlow,"Proc. Proceedings of the 2010 internet network management conference on Research on enterprise networking,2010.
[5]Hassas Yeganeh et al,"Kandoo: a framework for efficient and scalable offloading of control applications," Proc. Proceedings of the first workshop on Hot topics in software defined networks,2012.
[6]Curtis et al,"DevoFlow: scaling flow management for high-performance networks,"Proc. SIGCOMM-Computer Communication Review,2011.