HopCon: A Home OpenFlow Controller Prototype

A study on implementing home network controller

Junchuan Wang
juw061@eng.ucsd.edu

Jiyu Yu
jiy116@eng.ucsd.edu

Junjie Lou
julou@eng.ucsd.edu

ABSTRACT

Home network environments lack tools to help home users manage, configure and troubleshoot their home networks. At the same time, the home networks itself is becoming more complex to satisfy different user needs. Network equipment vendors either provides specif solution that doesn't scale, upgradable or products that complex for users to handle. It is both difficult to configure the network according to predefined policies, and to reconfigure it to respond to faults, load and changes. Motivated by these challenges, we believe Software Defined Networking (SDN) is a potential way to solve the problems. SDN is an emerging paradigm that breaks vertical integration, separating the network's control logic from the underlying routers and switches, promoting centralization of network control, and introducing the ability to program the network. Here we propose a home networks management prototype, HopCon, to help home users manage their networks. We present both the design and implementation of HopCon with evaluations on the performance of its use in managing home networks.

General Terms

Management, Evaluation, Experimentation

Keywords

Software Defined Networks, OpenFlow, HopCon, Quality of Service, Home Network Management

1. INTRODUCTION

The last two decades have seen the steadily increasing of the home network bandwidth and the popularity of the domestic Internet access. Internet content is becoming increasingly interactive including applications like video streaming and online gaming which are more bandwidth intensive and latency sensitive. In the meanwhile, the home network itself is becoming more complex, integrating a range of networking hardware to satisfy the users. Home network environment has its own challenges compared to other network environments. Typically, home users do not possess the skills or willing to be burdened with the issue of home network management, configuration and troubleshooting[1]. Home networking will be totally different based on user patterns. Mechanisms will be needed in management, troubleshooting and performance optimization of home networking. Apparently, ISPs have the responsibility to support their users. However, they do not have the direct access to the devices. Motivated by the challenges stated above, we believe future home networks should provide two capabilities. First, the capability to virtually split network on a per-flow basis; second, the capabilities to allow independent programmatic control over the partitions. Here we propose home network challenges and approaches we are trying to tackle them.[2]

Isolation of bandwidth: Each device has its own request of bandwidth usage. Minimal utilization will be guaranteed. The switch should guarantee the devices will not affect each other. The utilization is always determined by control plane and applications have no idea about whether they are limited on their bandwidth utilization. However, QoS is also a concern. Flows with high priority will be served first. Also the switch should be able to dynamically change the bandwidth utilization to better serve the home network users.

Direct control: Control plane should resides close to switch to provide direct control on flows. It will also provide transient response from change of policy determined by the users. Control plane will multiplex the flows to desired device while providing bandwidth limitation depending on user needs. Placing the control plane close to the switch seems working in all these situations.

Customization and ease of use: Home users should be able to modify and customize the bandwidth utilization according to their specific needs. However, the complicated implementations should be hidden under the APIs to provide easy-use products. Home users always lack the knowledge and experience of networking. They will not want to know how the packets are routed. Thus ease of use is also a concern in the design toward home networking management.

Thus, we believe Software Defined Networking (SDN) is a potential way to solve these problems. SDN is an emerging paradigm that breaks vertical integration, separating the network's control logic from the underlying routers and switches, promoting centralization of network control, and introducing the ability to program the network. First, it

breaks the vertical integration by separating the control plane from the underlying switches that forward the flows. Second, with the separation of control plane and data plane, network switches become simple forward devices and the control logic is implemented in the contoller. This greatly simplifies the configuration of switches. The separation of control plane and data plane can be achieved by implementing a well-defined programming interface between the two planes. Then with appropriate use of the APIs control plane can directly control the data plane elements efficiently. OpenFlow is one of those APIs. With OpenFlow, a switch can perform roles not just a switch, but also a firewall, load balancer, etc. We implemented a prototype of an OpenFlow switch named HopCon. Its functionality includes traffic monitoring, bandwidth control, parent control, speed test and QoS services. It also has easy-to-use UIs to interact with users.

The remainder of this paper is structured as follows: in Section 2 we explain why we are motivated to use Software Defined Networking (SDN), particularly OpenFlow, to tackle the challenges we described above. Section 3 proposes the architecture of our design. Section 4 describes the design choice and consideration of tools. Section 5 focuses on our implementation using Ryu APIs to approach our goals. Section 6 uses utilities like ping, iperf, tcpdump to evaluate the performance of our HopCon prototype. Section 7 reviews related work, and finally Section 8 discusses future works and Section 9 concludes.

2. MOTIVATIONS FOR THE STUDY

With the inspiring development of SDN eco-system, the Internet is now being re-factored by programmers. With the separation between data path and control flow, the software layer can be decoupled with the underlying hardware if the hardware provides a general interface. Our study is motivated upon the observation that most home network still uses vendor specific network operating software. They bought router or switch with controlling package pre-installed inside firmware. We envision a future that with OpenFlow, the controller needs not necessary vary due to the choice of vendor. Thus the controller can be adaptable to many devices as long as they support OpenFlow. This should bring easier configurations and deployment for the network. Also with community support, more sophisticated programs or software could be written, shared and distributedly deployed with little or non modifications. OpenFlow, as a new network protocol, and a new open standard, currently has more applications on industrialized network or production network. Deploying it on home networks is still at research level, we want to find out the logic if it should, or should not be widely used, what can be achieved to manage a controller through OpenFlow standards.

We define the objectives of our study as follows, we first build a controller prototype and tests its functionality in simulated environment. We would like to see how a development process based on OpenFlow would look like to build module that split flows, monitor/shape traffic, and control the network at flow level. With our own QoS module done, we want to know at what extent of flexiblity could OpenFlow bring in terms of QoS policy making.

Before starting the project, we found similar topics in [1], [3], [4], [5], [6], and [7], but later in our project, we implement the controller from our own point of view and injected our thoughts when consider details. When trying to implement a feature, we do not necessarily find the optimal solution in the industry, but want intuitive OpenFlow-based solution. Our project proves the feasibility of those objectives above. Finally we have a working prototype that meets the basic requirement and is highly extensible.

3. ARCHITECTURE

We first consider the high level architecture of the components. We first locate the controller in the network, and then discuss what should be the abstraction to manage the network resource.

3.1 Locating Controller

Given the considerations we raised in the motivation part, first we need to decide where to put the controller. We first considered how traditional home routers put the controller, i.e. the software or firmware within the chip shipped. So one idea is to put the controller upon switches. Since controller is just like any other binaries executes. So if a switch can run the controller as a process, then the controller would be a specific piece of software. Some advantage we think of this plan might be: (1) possibly lowest delay between control layer and data path. (2) Better synchronization in state propagation. The consistency is better guaranteed. Even if the switch fails, the controller would know very fast. What is more, they possibly fail in some same conditions. The drawback is pretty obvious. It limited the multiplexing of the controller resources. Each switch at least needs a controller instance, if not a brand new controller that is specifically designed.

We call the second schema we came up with "user-local". That is to put a reference controller on the switches, which is the basic functionality of routing as a router. The controller provides an interface that it can run any user application as the switches' controller upon their request. So user 1 can run his controller, and when he disconnected and another user 2 came up, that user 2 can "plug-in" his control logic. But we immediately spot a drawback here. It still requires central management logic. Because a central scheduler might need to deal with multiple requesting controller apps and see if those requesting apps are conflict each other. Or maybe it will just use control logic one at a time. If a controller must do the scheduling, we probably can expect it to do many more things such as version control and maintenance for controller software.

So this final architecture we deployed is to place to controller remotely on a cloud. We further assume that cloud is managed by an Internet Service Provider, because ISP is the closest portal the network which also connects home network. Since it directly provide bandwidth to the home network as end user, and is the "hub" of multiple end home networks, it is the perfect place to hold the controller. By definition, the cloud infrastructure is supposed to provide multiplexing by virtualization. So we can image such a scenarios: A controller base can instantiate different controllers on demands for home networks to use. Also the end users can upload their application to the cloud space. We found that this idea also relates to the management out-sourcing concept

mentioned in [8], in that paper, the author discussed cases that home networks want to outsource the management and operations due to security concerns and leverage the generality and programmable of SDN to achieve better security management. They also points out that due to the distributed network monitoring, the controller is more capable of preventing an attack. We don't intend to bring the distributed property into our context, but we feel the same as the paper that outsourcing can achieve expert consultancy. It also has the easier updating/upgrading advantage mentioned in second scheme. But there is an obvious tradeoff, that is the fact that flow set-up time might be increased due to the delay from forwarding packets between local switch and the remote controller on the cloud. Notably that this is only a one time overhead, which could be ignored here because home network are generally stably topologized.

3.2 Layer of the Hierarchy

In a top-down hierarchy view, the bottom level network components are the nodes and links. In OpenFlow's context, the resource to be manipulated can be as fine as a flow. But it might be very difficult to achieve some goals by controlling flows. A upper layer abstraction is thus introduced as flowspace[6]. A group of flows can be treated as one element to apply policies without difference. Furthermore, tools like FlowVisor can divide the control flow and link them to different controllers. It has its application on production networks. But here we argue that we might need to inspect if they are really useful or will just bring extra overhead. Inarguably, one might conclude from the discussion above

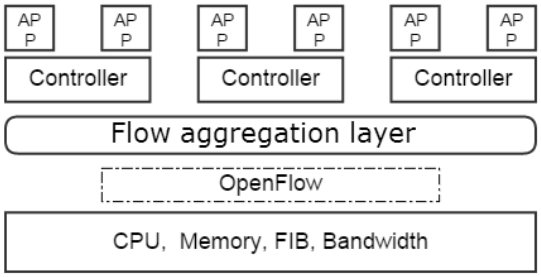

Figure 1: an additional layer for control virtualiation

that finer-grained subdivision for the network could be quite beneficial. So just like the case we mentioned in the second scheme. It is hereby possible that user 1's policy and user 2' policy can be run separately and quarantined in their sub networks domain, this layer of flowspace is a light-weight alternative of sub network virtualization and can support those policies to co-exists even they conflict with each other. OpenVirtex[9] also has been developed to provide true network virtualization. But we don't intend to do provide this level of granularity right now, because we suspect an extra layer of indirection would bring overhead by hiding implementation and providing transparency. Also we don't need to consider the scalability, so we are going to put them in the future work.

Another reason is that simply that we don't have tools to verify if another layer would bring such overhead as we suspected: FlowVisor lost supports from its developers from early 2013 and it does not support OpenFlow version higher than 1.2. Also the OpenVirtex, a promising alternative of OpenFlow, cannot only provide control virtualization like FlowVisor did, without simultaneously providing addressing virtualization and topology virtualization. Besides, it only works using Oracle JDK and is known to be buggy if using OpenJdk.

3.3 Topology and its Simulation

The figure below is an example home network topology has its controller setting up on ISP end. During our project we have built such a prototype controller that remotely control the switch. Users are assumed to be non-experts and therefore they interact with the controller using a Webpage-based user interface. So the User, Web-UI, Controller, switch has formed a communication chain. This is the topology that

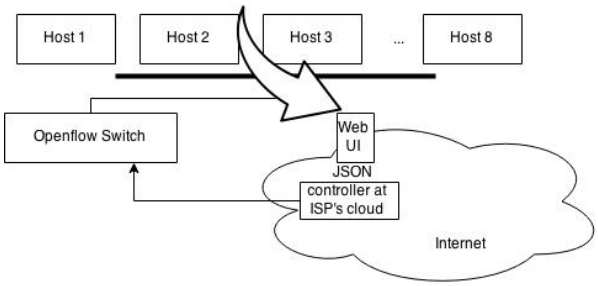

Figure 2: Example topology of home and controller

we depicted. During this stage, we also considered how we are going to simulate this topology once the controller is implemented. Here we use Mininet 2.2.0 as the test platform, which support OpenFlow up to version 1.3. We run it on Ubuntu 14.10. For the virtual switch, we use the Open Vswitch that comes as part of Linux Kernel. We can create such a topology in Mininet. We use two switches S1, S2

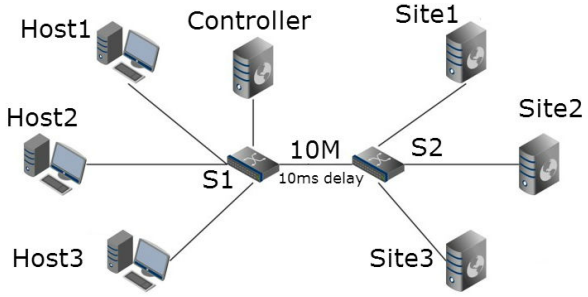

Figure 3: Experimental Mininet topology

here because we want to simulate the network bandwidth provision by setting configuration parameters between S1 and S2. From the figure, we set it to 10 Mb to simulate a 10 Mb incoming bandwidth provision. The controller is controlling S1 only. We also build three hosts to simulate host server of a website, as denoted as site1, site2, site3 here. It is worth nothing that since all hosts and switches are actually running on the same hosting machine (the one Mininet is running on), as a result, a weird implication is that controller, Web-UI's server, the switches (S1, S2) are

all addressed as LOCALHOST (127.0.0.1), with exceptions that hosts are assigned IP addresses by Mininet. To better simulate the topology, we need to separate those host and switches to their own network namespace and connect them using virtual interface pair, and then attach them to OVS-bridge. Then they can be assigned separate IP addresses in sub net to controller and switches to simulate the real topology.

4. DESIGN CONSIDERATIONS

In order to implement a controller, we decide to use a framework to accelerate our process, so that many lower layer work can be saved, for example, the controller communicate with switch using a certain type of protocols. In OpenFlow specification, it is OF-Config protocol and for OVS, the switches communicate using OVSDB. A framework encapsulates the communication module between the tools so users can directly call APIs within the program. There are a few choices and we choose Ryu[10], which is written in Python and is a lightweight framework. It has high community support and has supported up to OpenFlow 1.4. It has been deployed on various occasions including OpenStack cloud management. Some built-in features, such as the http server, can provide us alternatives for calling APIs. For the communication between controller and switch as aforementioned, instead of calling APIs, one can use curl command to POST or GET data using JSON. It is highly compatible with web UI. For web UI we use Javascript on HTML and use Jquery and Ajax to make the best of easy JSON processing.

For the OpenFlow version, we decide to use OpenFlow 1.3. Since the major release of OpenFlow 1.0, each release [11] aim to enhance the data structure or extend to protocol to improve the QoS provision ability. For example OpenFlow 1.1 extend flow action to instructions. OpenFlow 1.2 added queues. OpenFlow 1.3 added meter tables and OpenFlow 1.5 extended Meter actions. The meter table has been described in [11] and since its being introduced in 1.3. It became a powerful tool than queues and flow entry to achieve QoS functionality, simply because it can match more data flows and provides more differentiated actions. We cannot use version higher than 1.3 either since OVS only partially support version higher than 1.2 and Mininet doesn't support higher than 1.3.

5. IMPLEMENTATIONS

Next we can implement such a controller to achieve the functionality that can solve the problems we defined. There are many ways to do a same thing. So for those requirements, such as QoS control, parental control and resource allocation, different vendors came up with various ideas and solutions. Some companies did them all the time, with scope as large as a production network or even as small as home networks. But again, as our design philosophy suggested, we need to leverage OpenFlow. So we try to implement those functionalities in an "OpenFlow" fashion. At the very end we will test the functions and give our evaluation of the outcome. They might not be optimal, and sometimes we cannot compare with other vendor's solutions. But we hope we can investigate partially and do some research to get the sense of which kinds of control that OpenFlow is capable with, which ones not.

5.1 Web server

One advantage of using Ryu is that Ryu has a Web server functionality corresponding to WSGI. Using this functionality, it is very easy to create a REST API for the front-end to use. Also, registering the controller to the web server will provide web server the ability to interact with the controller.

5.2 Traffic Monitoring

The switch contains the traffic information for each port and each flow. Normal it's hard if a user wants to access these information direct from the switch. SDN provides us a way to easily deliver these information to the user.

5.2.1 Design Flow

First, we open a thread called traffic_monitor in our controller, the task of traffic_monitor is to send a OpenFlow *OFPPortStatsRequest* message to the switch. And also add one event handler to deal with the *EventOFPortStatsReply* message sent from the switch. Second, when switch receives the request, it will reply with the port information in Json. Then the *EventOFPortStatsReply* event handler in controller is triggered, parses the Json data and stores the information in memory since data is very small. Third, when user wants the port information, web browser send a request to our web server and web server directly access the controller to get data and reply it in Json to the web browser. Last, upon receiving the data from web server, browser will display this information to user.

5.2.2 Trade-offs

There are trade offs between accuracy and overhead. The traffic_monitor is poling information from the switch, and calculate the per port speed base on the difference of old and new *rx_bytes*. So we can increase the accuracy if we poll the data more open, but the cost is an increase of workload for both the switch and the controller. The objective now becomes get the acceptable accurate data with minimal cost, in our design, we set the polling time to 10 seconds.

5.3 Parental Control

Implement parental control using SDN can be more powerful and convenient than parental control in each device. Since switch have the global knowledge for all the devices in home side, and it's more secure because there is nothing device can do if switch already block the packet it want. What's more, parental control can be bi-direction: we can block the access to certain web server, we can block any device in our home to access the Internet, or prevent certain devices to access certain website.

Step 1. User send the request with the URL of the website or the IP address of the device he/she wants to block.

Step 2. Web server receives the request, check if the data is URL or IP, if URL, get the IP address of the URL from DNS, we create a pseudo DNS in the server to simulate the real DNS server.

Step 3. Web server adds this IP address to the controller's black list, and ask controller to add a flow with *dst_ip* equals to the IP it want to block. Also set the flow with a high priority and no actions. This is because in openflow 1.3,

there is no action that can tell the switch drop the packet, a flow with not action will drop the packets belongs to it. And since the priority of this flow is large than normal, even if same flow already exists in the switch, we still can block the packets whose destination IP is the IP we want to block.

Step 4. If the user wants to unblock the website or IP, first, web server ask the controller to remove this IP from the black list. Then the controller will send a *OFPFlowMod* message with the *OFPFC_DELETE* action that matches the *dst_ip* to the IP user want to unblock.

5.4 Bandwidth limit

For home networking, bandwidth limit is always referring to the downloading bandwidth of the devices, since most upload traffic is web request and Ack, which are usually very small. To limit the download bandwidth of one device, we add a queue in the switch to the port connecting that device. *OF-Config* is a protocol for the management of the OpenFlow switch, it has been defined as the NETCONF (RFC6241) schema and can perform status acquisition and settings of logical switch, port, and queue. Ryu provides us simple interface for Queue setting, when web server receive the bandwidth limit request, it parses the data and then wrap the data to be queue setting request and then send it to Ryu. This request contains both the port information and max-rate, and the queue is set to be queue 0 of that port, which is used by normal flows. More advanced Queue setting is discussed in QoS section.

5.5 Speed Tests

Just like parental control, we would argue that switch is a better place to carry on speed test than any other hosts. Now we have ever evolving speed test technique being developed on the desktops. We might just open a browser and then have our speed tested using HTML5. Even ATT has the speed test service on its official websites. When the network is flowing freely, those different tests would just render satisfactory result to the same extent. But when the network condition deteriorated, the end user would probably only get a bad result without knowing where to further inspect. However the switch might possibly find out what is really happening. In this sense the switch speed test provides statistics for diagnosis. If certain algorithms or protocol is running by the switch, it is possible for switch to find out the real broken link by doing control experiments. So we expect the speed tests conducted on switch is as less coupled as possible. Since the switch is the one manage all connections, it is easier for the switch to set up a ideal environment to conducts control experiments.

Our suggested testing process is conducted as followed. Suppose a user requested a speed test to test the real-time bandwidth provided by ISP, from the controller Web UI, and the switch responses and proceeds. At the very start, the switch would ping the host to make sure the link layer condition is good for this hop. If the performance downgrade at this stage, it is high possible that the segment connected to this interface is congested (such as broadcasting storm), or possibly in a wireless home network case, the AP that connected to the switch interface is congested. The performance drop is detected in a lower link-layer level. On another hand, if this ping performs better than certain threshold. Then the switch should proceed to testing ingress link speed.

The switch would then shut down all other session connections, and only establish one session between the requesting host and the remote ISP server. During this time all other applications' requests for connection on all other ports are blocked. Then say the user host downloaded a picture from the remote ISP server, tested speed is just the current bandwidth on the link that connect home network to the ISP's end router. If that bandwidth is lower than ISP promised, it is likely the ISP's egress link is congested on its router's port. The correctness of this methodology is based on another assumption and rule we set: the switch should be able to using "match" to detect all the UDP packets coming in the home networks. If it detected overwhelming UDP packets to congest the ingress link, the switch should know this and report. So when the link is not congested by UDP, we have ruled out the factors that affecting the bandwidth on the switch side and should be able to get correct bandwidth.

This is just an example for speed test for simple single model case, experienced network administrators can similarly write their testing logic to control switches in more complex cases, different topologies to help detect and diagnoses network problems.

5.6 Quality of Service

QoS on various context, can take place in various layers. They are implemented everywhere. They can be implemented in Ip networks, MAC layer on 802.11 networks[12] or wireless Networks[13]. But in general, it usually means differentiation on admission control, bandwidth reservation, delay settings for a certain type of traffic.

In OpenFlow, possible tools for QoS are queues and meter tables. A queue is a data structure buffering packets, and is associated only with one port connecting to that link. A port can have multiple queues, the actual number of the queue depends on switch implementations. As a result congestion control is achieved by setting different maximum rate and minimum rate on the queue. The QoS would be setting match between flow and queues, and gave the rule priority number, if necessary to provide preemptive feature.

A major downside of the queue is that queue only controls egress traffic. Say in the following topology we want the egress traffic does not exceed 1 Mb, we can route the traffic from h1, h2, h3 to a queue that has maximum limit of 1 Mb. However when the direction changes, we cannot control the

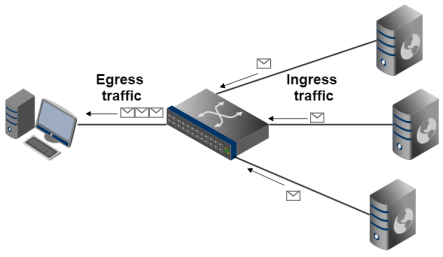

Figure 4: Egress and ingress traffic

real-time traffic that goes into h1, h2, and h3 to sum up below 1 Mb. All we can do is maybe set a queue for each of the port connects to h1, h2, h3 respectively, and then set the sum of their maximum possible rate to be below 1 Mb. Add another switch would solve this problem, but we assume there is only one switch in the network. *Another* problem here is the observation results from experiments that queue can only provide guarantee per user sense, instead of the one we want, i.e. per flow sense. For example, if User1 want to provide 1 Mb guarantee for its UDP service using port 5000, provided the link capacity as 10 Mb, he can match those traffic to the outgoing queue with min-rate. Then with other traffic unchanged, the UDP traffic associated with port 5000 will get the guarantee. However, what if User 2 participates here want guarantee rate of 8 Mb. Even though the sum doesn't exceed the maximum of the link capacity 10M, we found that the proportion they got is quite random and often is the case that User 2 never get that 8M guarantee while User1 get much more than 1M. We studied on various references, experiments and conclude that the OpenFlow cannot schedule to resolve the contention between User1 and User2. All he can do is to provide guarantee when there is only one user.

For the first problem, meter table came to salvation. Since OpenFlow 1.3, the data structure meter table is introduced to monitoring packets statistics match to that table and we can have action options on them. It can process ingress traffic and match them all to one table. So in pervious example, we only need to match the traffic go into h1, h2, and h3 to a meter table and set the maximum rate. For the second problem, the only solution is switch-specific supports. Apart from these two problems, We cannot do content-based service differentiation either. Because for the popular frameworks such as IntServ, DiffServ, MPLS,etc), they all leverage header information, so extra routers needed at edge router or transmission side to mark the header for fields like dscp. Our home network topology lacks a marking/labeling mechanism, since there is only one router.

So we leverage queue and meter table to implement traffic shaping for the flows. The granularity here is the flow. Using the bandwidth control techniques mentioned above, we can limit the bandwidth of a user, or a flow. When we set the flow to match things such as "UDP, port 5000", we can have a specific service difference. When we set priority number, we can achieve bandwidth preemptive. Based on the considerations above, we define our three QoS functionalities as rate proportional allocation, best service guarantee for certain service regarding a specific user and service differentiation based on whether service is TCP/UDP with port number. Those functionalities, or QoS rules, can be further customized by making combinations of different port number and protocol parameters. In the real world, we expect this kind of rules to enhance the service experience. For example a user may want to set their SSH service to be high priority and is provided minimum guarantee, he can achieve this by setting TCP/UDP service on port 22 with advantageous parameters.

6. EVALUATION

This evaluation section will evaluate the function we achieved during the project, as well as the OpenFlow controlling mechanism itself. We will further evaluate the controlling ability and programmability of OpenFlow, which lead to conclude the question that drives our study, and that is whether OpenFlow will be a good choice to make home networks easier to manage.

6.1 Problems during Implementation

We meet several problems during our implementation, most of the reason is because SDN is still a relatively new area, the software simulation tools are still evolving, and lots of new features are added but not stabilized. We believe this situation will be gone in the next few years.

6.1.1 Meter Table Support

The first problem we encountered is the lack of support for meter table from Open vSwitch. Meter Table is introduced in the OpenFlow 1.3, makes it enable to use policing of traffic in OpenFlow mechanism. And it can be useful to us to provide more flexible QoS service, but we tried different version of Open vSwitch, from 2.0.2 to the newest 2.3.1, through OpenFlow 1.3 is already supported in OVS, but the meter table feature still not supported. The solution to this problem is OpenFlow soft switch (ofsoftswitch). But ofsoftswitch has its own drawback.S

6.1.2 Ofsoftswitch

Ofsoftswitch can support meter table, which is good, but during evaluation, we found out that the TCP traffic sent by iperf will crush the switch. Later we found out the problem is caused by packets bigger than the MTU size as the designer of ofsoftswitch claimed. So we haven't evaluation the functionality of using meter table control TCP data.

6.2 Evaluation Scenarios

Functionality. We want to test the functionality of our design, how queue and meter table's performance as for the bandwidth control, and will QoS service provide the service we want to achieve. **Overhead.**There must be some kind of overhead when introducing a remote controller on top of a switch control the behavior of that switch. We tested the overhead of setup a flow when one host wants to ping an unknown host. We also measure the overhead of adding queue to the switch by testing the Average TCP packet RTT time between two specific hosts.

6.3 Tools

We use the tools that Mininet already provided to test the functionality of our design as well as the overhead that controlled introduced.

ping.ping can test the connection between as hosts, we also used ping RTT to test the overhead of setup a flow in the switch.

iperf.iperf can be used to measure the TCP/UDP bandwidth of two hosts, we use iperf to evaluate the functionality of our bandwidth control as well as QoS service.

tcpdump. We use tcpdump when we meet unfamiliar problems during the implementation, also be used to gathering packets information for future analysis

tcptrace. Tcptrace is used to analyse the packets statistical information like average RTT time.

6.4 Test Methodology and Results

6.4.1 *Flow Setup Time*

To test the overhead of setting up a flow, we let one host to ping another unknown host, and we use the first RTT time of that ping to approximate the flow setup time, we run this for 20 times for both local controller and remote controller. And every time we reboot the Mininet in order to flush the flow table in the switch. As we can see from the Fig. 5, the flow setup time of the remote controller is 14.8ms longer in average. The reason is because for remote controller, whenever flow miss is detected, packetIn is triggered, the packet will be sent from the switch to the controller for further action, the flow is added to that switch when switch receive the add flow message from the remote controller.

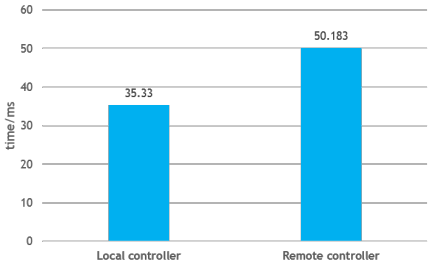

Figure 5: Flow setup time

Fig. 6 shows the RTT of two hosts when flow already added. As we can see, after flow added to the switch, there is no extra overhead of remote controller, the mainly overhead only exists in set up time.

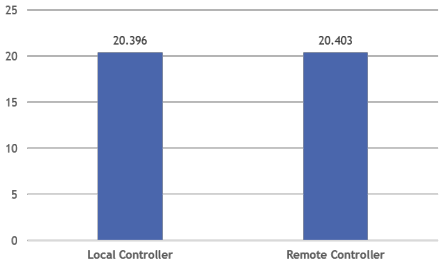

Figure 6: RTT after flow set up

6.4.2 *Queue Overhead*

We use the TCP packet RTT time to measure the queue overhead. Iperf provide us a way that can conveniently send TCP traffic between two host, we set up a iperf server and let one iperf client send TCP traffic to that server, and we open tcpdump at the sender's sides. After transmission finished, we use the tcptrace to analysis the packets information. Repeat the same work before and after we add queue to the receiver's port. We collect 439 samples be queue adding and 433 samples after queue adding. The result is in Fig. 7.
The average of RTT time with queue is 7ms larger than the RTT without queue, a very small percentage of the total average. The large RTT is due to congestion when using iperf.

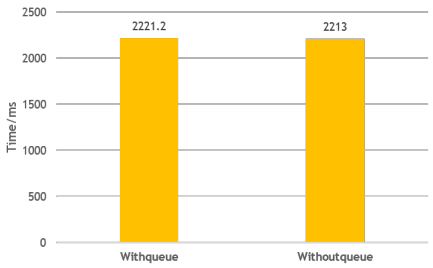

Figure 7: RTT time w/o queue

Queue ID	Max-rate	Min-rate	Port
1	500Kb/s	-	1

Table 1: Queue configuration

But if we extract the packets before congestion started, the average RTT is 21.9ms and 20.6ms. The difference is acceptable since 1 ms can be added anywhere in the network. Our conclusion is that when dealing with large flow time critical service like video games where several ms can make large difference, add queue in the switch is not suggested, overviews, queue is a simple and efficient tool to solve the bandwidth control problems.

6.4.3 *Queue-based Bandwidth Control*

This test will test how fine granularity queue can be when limit the bandwidth. First we add queue to port 1 that connected to host1, set the max-rate of the queue to be 500kb/s, then we open an iperf server on host 1, iperf clients from h4 and h5 and start to test the bandwidth of host1. The result is Fig. 8.

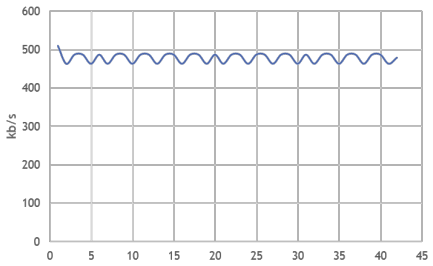

Figure 8: Bandwidth control using queue

The bandwidth is very well controlled, its a little bit smaller than our setting, almost at 500Kb/s, through there is a small oscillation of the bandwidth, indicated that queue based bandwidth control can not set the bandwidth to a strict fixed rate, but the result is totally acceptable.

6.4.4 *Meter Table based Bandwidth Control*

Then we used Meter table to control the bandwidth of the ingress traffic of the switch. And we set a QoS rule that match any UDP traffic go through the switch, combine this rule with the meter table we set. And also open an iperf

Meter ID	Flags	Bands
1	KBPS	type:DROP, rate:400000, prec_level:1

Table 2: Meter table configuration

Rule ID	Match	Action
1	nw_proto: UDP, dl_type: IPv4	meter:1

Table 3: QoS Rules

server at host1, a iperf client at host2 sending UDP traffic to host1. The result is in Fig. 9.

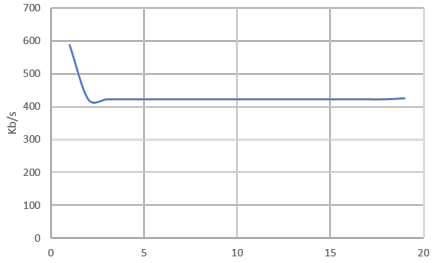

Figure 9: Bandwidth control using meter table

From the picture we can see that meter table control the UDB bandwidth to be exact 423Kb/s, 23Kb larger than our setting be at a very stable level, considering that the type of the meter table is DROP, which will drop the packet if bandwidth if UDP packets exceeds the limit. Unlike TCP congestion control method, we can avoid the little oscillation. Fig. 10 is the result of two hosts send UDP packets to two other hosts, due to the propriety of meter table ingress control, the total bandwidth of the two hosts combined is limited to 434Kb/s.

6.4.5 QoS Rules

We use queues and meter table implement several QoS rules that user can choose. In this section we evaluated the one of the QoS rules–traffic proportional. Users can set ratio of bandwidth between different hosts. Initially, not ratio is set and all three hosts get fairly equal bandwidth during congestion, after 10s, we apply our QoS rules to the switch and set the ratio of the bandwidth to be 5:1:1. The result is shown in Fig. 11.

The figure shows that, before our QoS rule applied, three hosts get almost the same bandwidth 1/3 out of 10Mb/s total, at 10 seconds, we apply the QoS rules, and the bandwidth of each host start to oscillate dramatically for about 10 seconds, after then the bandwidth start converge, host1 to average 6.3Mb/s, host2 and host3 to 1.34Mb/s, host1 is 5 times larger than host1 and host2.

7. RELATED WORK

Calver et al.[14] proposed a design for universal logging platform. They argue one challenge in managing and troubleshooting home networks is that knowing what is happening. Availability of a record of events that occurred on the home networks before trouble appeared would go a long

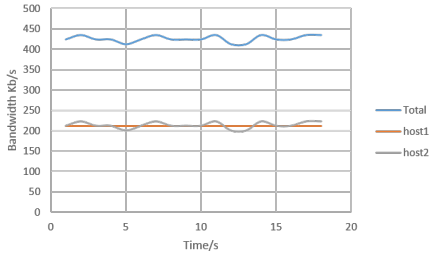

Figure 10: Bandwidth control using meter table with two receivers

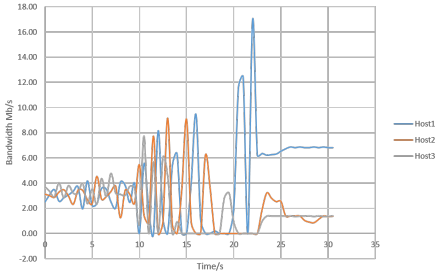

Figure 11: Bandwidth of each host before and after the QoS rule is applied

way toward addressing that challenge. They claim that an autonomous, comprehensive, and scalable logging platform is a key component for many interesting services related to home networks. The facility supports not only troubleshooting services for home network users but also other interesting applications.

Yakoumis et al.[2] study the systematic problems of current home networks. They argue broadband networks are expensive to deploy and lack the accountability of several different service provides. Also the home network is becoming more complicated because of connection with more devices, new applications and different user needs. In the meanwhile, the authors claim that there are no mechanisms or technology in place allowing ISPs to manage and improve the home network. Therefore, they propose slicing home networks as a way to overcome the problems. Slicing allows multiple service provides to share a common infrastructure and support many policies for cost sharing. They propose four requirements for slicing home networks: bandwidth and traffic isolation between slices, independent control of each slice, and the ability to modify and improve the behavior of a slice. They evaluate how these requirements allow cost-sharing, out-sourced management of home networks, and the ability to customize a slice to provide higher-quality service.

Fratczak et al.[1] provides a major influence to our work. They proposed HomeVisor, a novel remote network configuration and management tool. HomeVisor is designed to enable remote home network troubleshooting and management. It supports fine-grained configurations closely suited to user requirements and provides a user agnostic and non-application approach. They evaluate HomeVisor's ability to outsource control to an entity outside the home network. In

particular, they focus on evaluating whether outsourcing the management of home network slice is viable, and assess the scalability and efficiency.

8. FUTURE WORK

For immediate following work, we can continue build modules, or testing current modules on a real networks instead of simulating it in Mininet. We might measure the extra overhead that Mininet may have optimized or overlooked. For the modules, we can extend to implement more. Here in the project we haven't done Firewall module. Because we assume the home network is self-constrained and may not need a powerful firewall. But this is arguable if one want to extend his network with more interaction with public network. So building firewall or a general security module would just bring other considerations to evaluate OpenFlow from another point of view.

We previously mentioned that FlowVisor can be used to group or divide the flows on a link and provides control for each separate flow. When we look back, it seems a good choice to do. From our implementation experience, a flow is the best suitable and probably the smallest best-fit level for OpenFlow to put control over. If the developers want to aggregate flows and apply logics upon a group of flows, they might need to either define new semantics, data structures or write complex software logic with really well-maintained design pattern. So an extra level of abstraction like FlowVisor did should be able to cater more complex requirements.

If a controller can maintain a backup storage, we think that can be utilized to realize more functionality. For example, we can set logs on the storage, then for QoS requirement like "block traffic starting from 8:00 am to 12:00pm", we can maintain a task list that will not be scheduled immediately. That will help relax many logics the controller program needs to do, we can poll this list every now and then to manage scheduled tasks. Also the controller can store traffic statistics to do the analysis, or optimize network performance by storing DNS caching. Also since our controller is running on the cloud, different controller's information can be combined for big-data analysis.

Also during our project, we feel it necessary to define the specification of semantics or language for QoS requirements, especially in home network. Users in home network may have more general requirements, instead of explicit, well-defined policies. We also argue that when multiple policies involving one resource element are applied, reverting one of those policies can possibly lead to consistency issue. So for QoS module, we envision an SQL like system that can process policy language and maintain consistency for concurrent requirements. We found related research ongoing like that in[15]. There are not so many alternative or details exposed. We feel this topic can be further delved.

9. CONCLUSIONS

In this study we did implement a working controller for home network. We achieved all our original goals except the content-based QoS. We show that OpenFlow reaches our expectations to achieve network management on flow-level. It is powerfully industrialized and thus can be easily tailored to achieve home requirements. It still has notable limitations. (1) Its lack of scheduling ability still requires switch-specific support to reconcile inter-flow scheduling contentions. (2) We cannot do content-based packet differentiation using only one switch since the switch differentiate packets through its header. So an extra marking router is needed. (3) The setting up overhead may be huge due to the packet-in process, which happens during the flow set-up stage.

As a protocol itself, OpenFlow cannot solve all the problems, but our conclusion is that for the functions it can do, such as managing flows in networks and forwarding packets on-demand, OpenFlow has provided a generalized solution which are efficient and bug-proof. We appreciate its extensibility and programmability and are hoping more vendors to provide OpenFlow-compatible router/switches.

10. ACKNOWLEDGMENTS

We are very grateful for our instructor, Alex C. Snoeren for his insightful and enlightening tutorials during the course, and Teaching Assistant, Bhanu Vattikonda for his extremely helpful instruction and prompt help. We also appreciate those people who kindly answered our question from Ryu Community.

11. REFERENCES

[1] T. Fratczak, M. Broadbent, P. Georgopoulos, and N. Race, "Homevisor: Adapting home network environments," in *European Workshop on Software Defined Network*, pp. 32–37, December 2013.

[2] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *HomeNets, ACM SIGCOMM workshop*, August 2011.

[3] H. Kumar, H. H. Gharakheili, and V. Sivaraman, "User control of quality of experience in home networks using sdn," in *Advanced Networks and Telecommuncations Systems (ANTS)*, December.

[4] H. H. Gharakheili, J. Bass, L. Exton, and V. Sivaraman, "Personalizing the home network experience using cloud-based sdn," in *A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June.

[5] I. Trajkovska, P. Aeschlimann, C. Marti, T. M. Bohnert, and J. Salvachua, "Sdn enabled qos provision for online streaming services in residential isp networks," in *Consumer Electronics - Taiwan (ICCE-TW)*, May.

[6] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer,"

[7] M. Chetty, H. Kim, S. Sundaresan, S. Burnett, N. Feamster, and W. K. Edwards, "ucap: An internet data management tool for the home," in *ACM Conference on Human Factors in Computing Systems (CHI)*, April.

[8] N. Feamster, "Outsourcing home network security," in *HomeNets, ACM SIGCOMM workshop*, September 2010.

[9] "Virtex." http://ovx.onlab.us/.

[10] "Ryu sdn framework using openflow 1.3." http://osrg.github.io/ryu-book/en/Ryubook.pdf.

[11] T. O. N. Foundation, “Openflow switch specification v1.4.0.”

[12] A. LINDGREN, A. ALMQUIST, and O. SCHELÃĽN, “Quality of service schemes for ieee 802.11 wireless lans âĂŞ an evaluation,” in *Mobile Networks and Applications*, pp. 223–235, 2003.

[13] Q. Ni, L. Romdhani, and T. Turletti, “A survey of qos enhancements for ieee 802.11 wireless lan,” *Journal of Wireless Communications and Mobile Computing*, vol. 4, no. 5, pp. 547–566, 2004.

[14] K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou, “Instrumenting home networks,” *ACM SIGCOMM*, vol. 41, pp. 84–89, January 2011.

[15] A. Voellmy, H. Kim, and N. Feamster, “Procera: A language for high-level reactive network control,” in *HotSDN*, August 2012.

APPENDIX

A. WEB UI OF OUR DESIGN