ASIC: An Architecture for Scalable Intra-domain Control in OpenFlow
Pingping Lin, Jun Bi, Hongyu Hu
Network Research Center, Department of Computer Science, Tsinghua University
Tsinghua National Laboratory for Information Science and Technology
4-204, FIT Building, Beijing, 100084, China
linpp@netarchlab.tsinghua.edu.cn, junbi@tsinghua.edu.cn, huhongyu@cernet.edu.cn

ABSTRACT
Currently, the architecture of network device is closed which is detrimental to network innovation. Software defined networking decouples the vertically coupled architecture and reconstructs the Internet as a modular structure. The idea of software defined networking is widely accepted. OpenFlow, a typical instance of the software defined networking, has been deployed by many universities and research institutions all over the world. With the increasing scale of deployment, the poor scalability of the OpenFlow centralized control mode becomes more and more obvious. To solve this scalability problem, this paper adopts the idea of load balance and proposes an architecture for the scalable intra-domain control named ASIC in OpenFlow. ASIC balances all the data flow initialization requests to several physical controllers in an OpenFlow network, and then, those requests are processed with a shared global network view in a parallel manner. ASIC also builds a data cluster for the global network view. By this way, the scalability problem in the intra-domain control plane could be completely solved. At the end of this paper, the emulation shows the feasibility of ASIC.

Categories and Subject Descriptors

General Terms
Performance, Design, Experimentation.

Keywords
Software Defined Networking, OpenFlow, Scalability.

1. INTRODUCTION
Currently, the architecture of the network device is closed. This is not favorable to network innovation, and protocols related to the core network lay or core network device are hard to apply. Software defined networking (SDN) [1] decouples the vertical and tight coupled architecture, and reconstructs the Internet as a modular structure. At the same time, it opens up the control plane and the protocol implementation in the control plane. Then, the architecture in network device is no longer closed. By this way, SDN promotes the rapid innovation and the evolution of the network. Currently the idea of SDN is well received by both of the academic researchers and industry researchers, and the SDN has become a popular topic in recent years and is considered as a promising way to re-architect the Internet. In addition, some research groups are formed such as the Open Networking Summit [2]. The Open Networking Foundation [3] is defining the standard for SDN, and more than 50 companies have joined the foundation to accelerate the creation of standards, products, and applications, such as Google, Facebook, Microsoft, IBM, VMware, Juniper, Cisco and so on.

At present, SDN already has some implementations such as Openflow [4] and NetOpen [5]. OpenFlow, the most popular instance, is deployed by many universities and research institutions in the world. OpenFlow adopts the centralized control model and is made up by the OpenFlow controller, FlowVisor [6], OpenFlow switch, and OpenFlow protocol as shown in the figure below.

Figure 1. Architecture of OpenFlow

OpenFlow separates the control plane from the network equipment, and moves it to the OpenFlow controller. The controller takes charge of all the functions in control plane, while the OpenFlow switch retains only the basic data forwarding function. The FlowVisor can achieve the network virtualization, and then, different applications on the network controllers such as multicast [7], network situation assessment [8] and network status statistics could form their own private virtual networks. Furthermore, the OpenFlow controller can just run in a normal host or server to control the data packet forwarding in OpenFlow switches through a standardized OpenFlow protocol and a safe channel.

In the OpenFlow centralized control model, all the routes are determined by the controller, so the first packet of each data flow...
is sent to the central controller. Then, the controller will compute the routing path for each data flow and install the routing path to the related OpenFlow switches according to a global network view. Here, the first packet of each data flow is usually called the data flow initialization request. In addition, the request processing capability of a single controller is limited: NOX [9] could process about 30K requests [10] per second; 2) Maestro [11] could process about 600K requests per second. In fact, large-scale network environments always have vast amounts of data flows: 1) a 1500 server cluster might generate 100K requests per second [12]; 2) a 100 switch data center might generate 10000K requests per second [11]. With the increasing scale of deployment, the scalability of the centralized OpenFlow control becomes more and more obvious. Thus, this paper proposes building a powerful logical controller made up of several physical controllers. Then, the scalability in the control plane evolves into a cooperative performance problem of multi-controllers as illustrated below.

**Figure 2. The scalability problem in OpenFlow control**

Based the analysis above, to improve the performance of the OpenFlow control plane, this paper adopts the idea of load balancing, parallel processing, data sharing and clustering to the OpenFlow network, and proposes an architecture for the scalable intra-domain control named ASIC: the first step is to balance the vast amounts of data flow initialization requests, and then process those requests by several OpenFlow controllers simultaneously according to a shared global network view. In which, each controller computes and installs the routing path for every request independently. In such way, ASIC has the ability to solve the scalability problem completely.

The rest of the paper is organized as follows: the next Section presents the related work. Section 3 describes the design of the ASIC architecture for scalable Intra-domain control. Then this paper gives emulation and evaluation to the proposed mechanism in Section 4. Finally, Section 5 concludes this paper.

### 2. RELATED WORK

The research to the scalability in OpenFlow control plane is mainly divided into two categories: 1) improving the request processing speed of the controller software itself. 2) new proposals by pre-installing the data forwarding rules to OpenFlow switches, or clustering the data flow initialization requests, or returning a certain right to the OpenFlow switch for local network events, or multiple geographical distributed controllers and each handles the local area OpenFlow switches.

#### 2.1 Research on High Performance Controller Software

At present, there are many kinds of OpenFlow controller systems: NOX [9], Maestro [11], Beacon [13], SNAC [14] and so on. Some researchers such as Beacon and Maestro are trying to improve the performance of the controller by the multi-thread technology. Furthermore, the controller software systems are always being deployed on a multicore host or server. However, for large-scale data centers or networks, a single physical controller is not enough which has already been shown in Figure 2.

#### 2.2 Research on Solutions for Scalable Control Plane

There already have some proposals for the scalable control plane, such as HyperFlow [15], DevoFlow [16], DIFANE [17], and ONIX [29]. HyperFlow proposes a distributed control plane for OpenFlow. In HyperFlow, a domain should run multiple controllers simultaneously, and each controller handles the local area OpenFlow switches. For sharing the global information among those controllers, HyperFlow adopts a distributed file system named WheelFS [18] which is designed for the WAN (Wide Area Network) environment. Each HyperFlow controller has the right to deal with network events within a certain local area, and the events which will affect the global network should be announced from time to time. Once other controllers learn the event information, they should replay the event to achieve the synchronization of the global view. However, this approach can only deal with events which do not occur frequently, such as link status changes. For some data centers or networks, the administrator needs to know all the real-time events which is not supported by HyperFlow at present.

DevoFlow adopts the clustering or fuzzy matching technology. After clustering for data flows, each of the controller processes is not a data flow, but a class of data flows. At the same time, DevoFlow gives the OpenFlow switch the decision-making right as much as possible. By clustering to reduce the burden of the controller, there is a compromise between the number of classifications and the burden of the control. The more loose the fuzzy matching rules are, the less the burden of the control. Conversely, when the fuzzy matching rules are strictly set to exact-match, the scalability problem will reoccur.

DIFANE returns some control right to the OpenFlow switch. By pre-installing several forwarding rules, once the matched data flows arrive, the OpenFlow switch does not need to forward the initialization request to the controller anymore and forward the flow directly as the pre-installed rules. In such way, DIFANE alleviates the burden of controllers and enlarges the control scope of a controller. ONIX treats each partition network as a logical node when make global routing decisions, and each partition network makes local routing decisions. The common drawback of DIFANE and ONIX is that they could not achieve the real-time visibility of the global data flow status.

By the analysis above, the existing solutions could only reduce the load of the controller to a certain extent, or increase the control scale of the controller by expensing the real-time network view. So far, there has not been a good solution to completely solve the control scalability for large-scale networks. In order to completely solve the scalability issue, this paper proposes a scalable mechanism with a three level architecture, in which each level has its own scalable approach to evolve.
3. THE ASIC ARCHITECTURE

3.1 Overview of ASIC

The capability of a single controller for handling the data flow initialization requests is limited. In order to deal with large-scale networks which have vast amounts of initialization requests, this paper suggests a multi-controller collaboration and network view sharing method to process the requests in parallel. ASIC is a powerful logical controller and mainly includes three levels: the first layer is a load balancing system; the second layer is the controller cluster; and the third layer is a distributed storage system for data sharing. In ASIC, the load balancer only forwards the requests, and the functions of each controller are equivalent.

When a data flow initialization request packet arrives at the logic controller, firstly it is forwarded to a physical controller located in the second level by the load balancing system in the first level. Then, the physical controller calculates the routing path by calling the global path based on the global network view, generating OpenFlow entries, and installing the entries to the corresponding OpenFlow switches. Further, for the replying routing path packet, the physical controller sends them directly to the OpenFlow switches without passing through the load balancer in the first level. In detail, the routing path is expressed by the OpenFlow entries in a flowtable [4]. The routing path installation is the process that the controller installs the flowtable entries into the corresponding OpenFlow switches by a secure channel described in the OpenFlow protocol.

3.2 Load Balancing to Initialization Requests

![Figure 3. Applying the load balancing to OpenFlow network](image)

As shown in the figure above, all the data flow requests within a domain are distributed to different controllers by the load balancing equipment, and the load balancer can do the requests distribution by a variety of algorithms like round-robin scheduling [19], or hash scheduling algorithm [20] by the IP address. All the physical controllers are equivalent. Their tasks are to calculate the routing path based on the global network view, generating OpenFlow entries, and installing the entries to the corresponding OpenFlow switches. From the prospective of a single controller, the role of the controller in ASIC is the same as that in the current OpenFlow environment.

In the actual network environment, the load balancer or packet dispatcher can be selectively adopted according to the number of requests. For example, networks might use a router as a balancer (using the policy routing model [21]: turning off the route learning function and forwarding packets according to the statically configured rules such as forwarding certain packets to a specific port), an OpenFlow switch (distributing requests by the action attribute [4] in flowtable), a professional Web load balancer (like the F5-BIG-LTM-8950 product of F5 corporation), or a software such as Click Router [22][23].

3.3 Storage Cluster for Data Sharing

In order to achieve the data sharing for the distributed control plane and to provide a consistent global view for each controller, this paper suggests adopting a mature data storage system which must include at least two parts: 1) persistent storage which must support transaction (A transaction is a group of operations, any operation in the group fails, the whole group must be undone or rolling back); 2) caching storage. Controllers directly deal with the caching storage in both reading and writing, and the persistent storage could be used for data analysis or maintain network status during a reboot process. We explain the data cluster in two steps.

**Step 1:** To give the readers a concrete picture of the persistent data storage, this paper takes the MySQL [24] database for example which could be replaced by other arbitrary persistent storage systems such as DHT (distributed hash table). MySQL is a mature technology with good scalability in both reading and writing. A feasible design for the data sharing in OpenFlow is as shown below:

![Figure 4. Writing and reading mechanism to the distributed storage of the shared data](image)

In the figure above, one master database might correspond to several slave databases. The master database is responsible for data writing (INSERT, DELETE, and UPDATE) while the slave databases are responsible for data reading (SELECT). The data consistency between the master database and slave databases can be achieved by the data replication mechanism. Thus, the scalability in data reading can be achieved by increasing the number of the slave databases. At the same time, to achieve the scalability in data writing, the master database should be split. A network administrator can vertically split the master database based on the actual requirement to form groups of one-master and multi-slave unit. For example, by grouping according to the source IP address, data with related source IP address is written to a same master-slaves database group. Meanwhile, in order to obtain a global view of the domain, the controller needs to read data fragments from slave databases of corresponding group units according to the actual requirement and, then, assemble the data fragments together, which is shown in the data reading process.

**Step 2:** Furthermore, in order to accelerate the data reading and writing speed, ASIC adopts a memory caching system (such as Memcached, [25] [26]). The memory caching system can cache data in the memory to shorten the data reading time compared to reading directly from the database. Therefore, based on the MySQL master-slaves architecture, to further accelerate the data reading speed, the data cluster in ASIC adds a memory caching system as shown below:
After adding the memory caching system to the data cluster, the data reading and writing against the persistent database are changed to against the Memcached pool. The Memcached pool and the persistent storage should synchronize the data from time to time. Then the persistent storage in the data cluster could be used for data analysis or maintain network status during a reboot process.

3.4 Scalability Analysis of the ASIC
The ASIC architecture can dynamically adjust its scale according to the number of initial requests or according to network size. The selection of the load balancing equipment can range from the software balancer to the current Internet backbone router. Thus, by using the backbone routers, the load balancing equipment will not bottleneck the entire logic controller. Along with the performance improvement from the load balancing device, ASIC can also increase the corresponding number of the physical performance improvement from the load balancing device, ASIC controllers, and different controllers could run different controller software systems independently. Once the number of requests increases, the number of databases should also increase correspondingly. Thus, each of the three levels in ASIC has its scalable solution and could be extended to meet the requirements of large-scale networks.

3.5 Application Deployment in Control Plane

The current OpenFlow experimental environment is always deploying one physical controller, which is unable to run a large number of applications. In ASIC, controllers behind the load balancer can run different controller software systems (network operating system) such as NOX and Beacon. The control plane applications from the third-party developers can be installed into one or several physical controllers. All controllers share a same global network view, and the only difference is the controller software installed. Thus, developers can selectively install their applications according to their preference of developing language or to the functions of different controllers.

4. EXPERIMENT AND RESULTS
To verify the feasibility of ASIC, this paper carries out the emulation experiment for the entire ASIC framework. The performance metrics of OpenFlow controller mainly refers to the request processing throughput and the data flow setup latency.

4.1 Experiment Environment and Design

Hardware configuration: ten hosts (Intel Core 2 Quad processor, 4GB memory, network interface card 1Gbps/port), five of which play the role of data flow initialization request packets generator. The other five hosts work as the Memcached cache servers. To ensure the requests sending speed, the request packets with different IP addresses are generated in advance. The load balancer in this experiment is a router with each network interface card for 1Gbps/port. All the controllers and the MySQL database servers are using blade servers (Intel Core 2 Quad processor, 4GB memory, network interface card 1Gbps/port), and each network interface card from any of the equipment in this experiment supports 1 Gbps/port. The switch is the Ruijie corporation product RG-S5760-24GT/4SFP switch.

Software configuration: all the hosts and servers are installed with Java-1.6.0_25 and the operating system Ubuntu-10.04. The controller software is Beacon-1.0.0 with mild modification for the experiment. Furthermore, each Beacon controller runs four threads. The cache server is installed with Memcached-1.4.13, and the database system is MySQL-5.1.42.

Experiment design: emulating an OpenFlow domain with the Multihoming [27] scenario in which two Internet service providers A and B are configured for the network export. Some of the users’ IP addresses belong to provider A, and some user’s IP addresses belong to provider B. When the logical controller receives request packets with different source IP addresses, it should use different routing paths to route them to the appropriate export.

Five requests generating hosts are connected to five different ports of the router, and the router cascades five controllers in the right direction. By such connection, the experiment emulates the data flow initialization requests process in a real network environment. In the initial stage, we first calculate out the routing paths and store them in the databases and then cache them in the Memcached server. By sending requests in high-speed to the
router, the requests processing capability of each controller could be counted.

### 4.2 Experimental Results and Analysis

#### 4.2.1 Throughput

By monitoring the Memcached server, the modified Beacon software indeed reads data directly from the cache server. By counting the number of fetching data from cache, the request process speeds of controller 1 and controller 2 are shown in the figure below.

![Figure 8. The request processing speeds of controller 1 and controller 2](image)

Further, we record the capability of all the controllers as follows:

<table>
<thead>
<tr>
<th>Controller</th>
<th>Speed Requests/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>414,598</td>
</tr>
<tr>
<td>2</td>
<td>403,470</td>
</tr>
<tr>
<td>3</td>
<td>425,553</td>
</tr>
<tr>
<td>4</td>
<td>383,376</td>
</tr>
<tr>
<td>5</td>
<td>394,511</td>
</tr>
</tbody>
</table>

In this experiment, the capacity of one single Beacon controller is about 400,000 requests per second. After the load balancing, the processing capability of the logical controller which includes 5 Beacon controllers is about 400,000 * 5 = 2,000,000 requests per second. In this experiment, we use only 5 controllers and one group of the MySQL master-slaves unit. If we expand the scale of the experiment deployment, the capacity of the logic controller will increase correspondingly in a linear growth. Thus, the purpose of load balancing is achieved.

The capability of handling requests is directly related to the size of the network topology. The larger the size, the longer the average time for the controller to calculate out a routing path is. In this experiment, each controller needs to compute for each request (according to the source IP address to calculate out which routing path to fetch) and fetch the global network view (reading routing path from cache), so it is reasonable of each controller to process 400,000 requests per second. With the expansion of the network size, we can do the vertically fine-grained division to the master database, and achieve greater data clusters and distributed storage. Thus, the data storage also has good scalability. If we adopt the Internet backbone router as the request distribution equipment (for example, the Huawei corporation product Quidway® S9312 switches can reach a throughput of 1,320,000,000 request packets per second in theory), in theory the logical controller can support the size of an actual autonomous domain in the current Internet.

#### 4.2.2 Latency Analysis

Comparing this powerful logical controller to a single controller deployed in the current OpenFlow network, the time delayed in ASIC is $O(h) + O(f(n))$ which could be accepted by end users based on the analysis as follows:

1) $h$ is the hops of balancers in the balancing system which a request needs to go through before reaching a physical controller. Because each balancer, like the router, has a limited number of ports to be connected by the physical controllers, cascading the balancers is necessary in some circumstances. Since each balancer is only responsible for the packets distribution, the time for traversing a balancer is equal to traversing an ordinary switch, so the time spent in this step can be accepted.

2) $f(n)$ represents the time consumed to fetch the global network view data from the shared data storage, and $n$ means the number of nodes where the data segments located. Because the storage is cached in the memory, controllers can get the data directly from the Memcached device in a high-speed. In addition, the distributed data can be fetched by multi-thread in a parallel manner. We suggest connecting the cache devices and controllers with the same switch. This way, the delay of fetching the distributed data can drop to $f(n/num(threads))$. Due to the two reasons analyzed above, the data fetching time $f(n)$ could be negligible.

### 5. CONCLUSION AND FUTURE WORK

Reviewing the development of the computer, the computer field adopted a common underlying hardware (x86 instruction set [28] based hardware). Based on the x86 instruction set, the computer operating system and software present a trend of diversification. Today's network is also showing a similar trend: OpenFlow reduces the logic of the network switching equipment and lets the data plane in OpenFlow switch to forward packets through a uniformed OpenFlow flowable. At the same time, the operation to the flowtable is through a public network API (Application Programming Interface) defined in the OpenFlow protocol. The control plane and data plane are decoupled and evolve independently. The control logics and the applications in the control plane are no longer relying on network equipment vendors to implement them, and the network researchers can do arbitrary innovations on the controllers. In such way, OpenFlow enables the rapid innovation of the networks and the rapid evolution of the Internet. Thus, OpenFlow can be considered as “the x86 instruction set of the network”.

This paper adopts the idea of load balancing, parallel processing, data sharing and clustering to solve the scalability problem in the OpenFlow control plane for the first time, and proposes an ASIC architecture: load balancing the data flow initialization requests and using multi-controller to process those requests in a parallel manner by sharing a global network view in a cached data cluster. Furthermore, in the ASIC framework, the three levels have their own scalability solutions and can evolve independently. By clustering the network device and building the large-scale data center, ASIC has the ability to completely resolve the scalability issue in the centralized OpenFlow control plane. The future work mainly focuses on deploying the ASIC to the actual OpenFlow network environment and monitoring its behavior and the actual performance.

### 6. ACKNOWLEDGMENTS

Supported by National Science Foundation of China under Grant 61073172, Program for New Century Excellent Talents in University, Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant 200800030034, and...
National Basic Research Program ("973" Program) of China under Grant 2009CB320501.

7. REFERENCES