Scaling of SDN: A new approach

Aniruddha Malkar

Abstract—

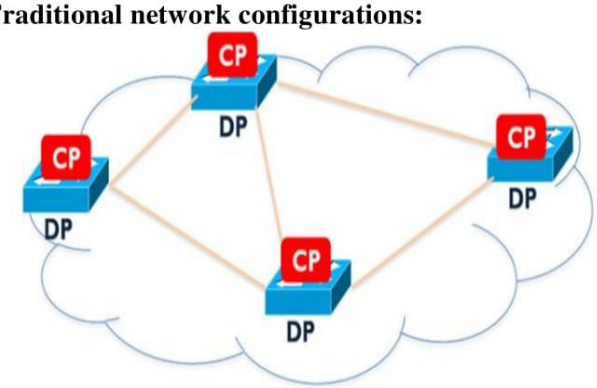
SDN moves the control plane of the network from switches to the centralized open interface controller such that the control plane is placed at software. The SDN network administrator configures the policies into the network with the centralized control. The SDN controller takes the responsibility to deploy the polices into the switches. The SDN provides great flexibility and improvement over traditional network such that the network can meet the current network demands. But the SDN faces scaling issues due to gathering of initial packets at the controller from all the switches. To mitigate such scaling issues the many solution were proposed but maximum of them don't have the comprehensive solution to mitigate all the issues. This paper revolves around one question that "Can we have a comprehensive solution to mitigate all the SDN scaling issues?". The paper proposes a novel architecture to mitigate all the SDN problems. The paper targets the embedded systems and it performance comparison with the conventional SDN controller. The results show that the embedded systems provides great improvement over the conventional SDN controller, which is a potential solution of the current SDN scaling problems. The simulated models are used in the paper and real implementation can be done in future.

I. INTRODUCTION

The IT industry is evolving at grate rate and continues to evolve with years. On the other hand, the networking field has limited innovation in last 2 decades. It results in complex and inflexible networks no longer meeting the current business requirements. At present business and technical network requirements includes improved performance and realizing the border connectivity. The organizations need to meet increasing industry specific security regulations and growing demand of mobility. In order to meet with such criteria networking protocols required to evolve significantly. But the traditional network setup makes deployment of protocol to realize such features is quite challenging.

The difference between the traditional networking and SDN is given below:

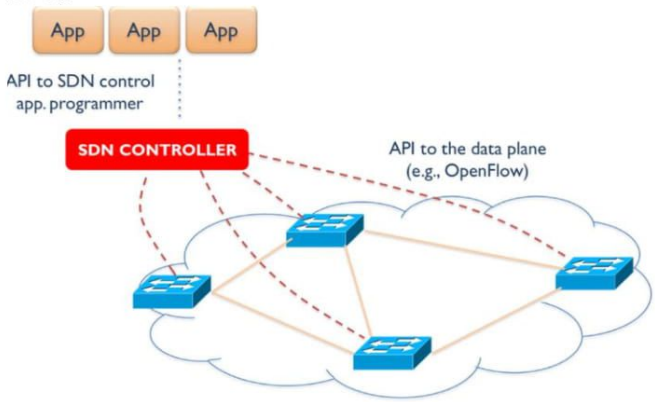
Traditional network configurations:



In the traditional networks, the network functionalities are essentially implemented in the dedicated appliances such as switches, routers, other network equipment's. Such functionalities within these appliances are implemented in dedicated hardware i.e. ASIC. Companies are increasing facing the resistance in improvement due to the limitations posed by hardware centric approach. The traditional configuration of such network appliances is time consuming and error prone. Multiple steps are required when the IT administration needs to add / remove single device in the traditional network. Following step is using the device level management tools to update various configuration settings like ACLs, VLAN etc. Such configuration approach makes the deployment of consistent set of policies hard to the network administrator, which causes the companies to encounter security breaches and non-competence with implications. The traditional interface induces high administrative hassles that hinders networks to meet with business networking standards. Multi-vendor environments require a high level of expertise. The IT organization owns numerous equipment of variety of vendors. To successfully perform the configuration, the administrator required to have the complete knowledge of the variety of devices present in the network. Traditional architecture complicates the network segmentation.

.

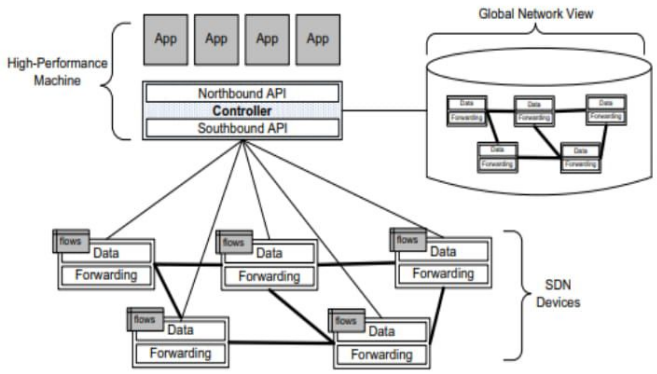
SDN:



(b)

SDN is becoming popular among the networking business. The emerging SDN technology will play significant role in overcoming the limitation of traditional networking. SDN decouples the control plane and data plane by leaving the data plane with the network hardware and moving the control pane into software layer. SDN abstract the network form the hardware such that policies are no longer executed in hardware but it uses the centralized software application functioning as a control plane. The SDN provides centralized view of the entire network and enables the centralized network provisioning. By abstracting the control plane and data plane, SDN can accelerate the service delivery and provide more agility in provisioning of virtual and physical network devices from the central location. SDN provides the complete enterprise management. SDN support the management of physical, virtual switches and network devices from the centralized controller, which is cannot be achieved through SNMP. The SDN provides the single set of APIs to make single management console for physical and virtual devices. The SDN controller can obtain information about the source requirement of application and resources availability at lower layers. The controller can able to match the resource requirement against the resource availability that result in efficient utilization of the networking hardware. The SDN architecture implement intelligence at centralized controller which reduces the complexity of network elements. By configuring the policies in the controller using OpenFlow the network administrator can implement the policies in a single API, the controller takes the charge of distributing and configuring the policies inside the network devices which relives the work of administrator and the policies are complete and securely implemented inside the network devices.

II. SDN OPERATION



The SDN architecture made up of three basic components SDN devices (switches/ routers), the controller, and the applications. The SDN devices incorporates the forwarding functionality to decide where to forward each incoming packet. The devices incorporate the data that drives the forwarding decisions, such data represents the flow defined by the controller. The flow is defined by the set of packets transferred from one network end point to another network endpoint. The set of rules defines the forwarding action that the device must take for the all packets from the same flow. Flow is unidirectional in that packet flowing between the same two endpoints, the opposite direction between the same two end point must consider as different flow. The flow represented by on the device as a flow entry. The flow tables are configured into the network device and consist of series of flow entries and the action to be taken when the packet matching that flow arrives at the device. When the SDN device receives the packet, it consults the flow tables in search match. Such flow tables are configured previously when the controller download the respective flow rules to the devices. When the SDN devices find the match, it takes the respective configuration action defined by the controller, it is usually entails with the forwarding packet. When the packets are not matched the switch forwards the packet to the controller. The SDN controller is responsible for abstracting the network SDN devices it controls and providing the an abstraction of such network resources to the application running over it. The controller enables the SDN application to define the flows on devices and it helps the application to

respond to the packet that are forwarded to the controller by the network devices. SDN applications are built on top of the controller, the SDN application are part of network layer two or three. The SDN application interfaces with controller with set of proactive flows on the devices and to receive the packet packets that are forwarded to the controller. Proactive flows are established by the application; typically, the application will set these flows when the application starts up, and the flows will persist until some configuration change is made. This kind of proactive flow is known as a static flow. Another kind of proactive flow is where the controller decides to modify a flow based on the traffic load currently being driven through a network device. In addition to flows defined proactively by the application, some flows are defined in response to a packet forwarded to the controller. Upon receipt of incoming packets that have been forwarded to the controller, the SDN application will instruct the controller as to how to respond to the packet and, if appropriate, will establish new flows on the device in order to allow that device to respond locally the next time it sees a packet belonging to that flow. Such flows are called reactive flows. In this way, it is now possible to write software applications that implement forwarding, routing, overlay, multipath, and access control functions, among others. The OpenFlow protocol as the means of communication between the controller and the device.

III. Issues in Scaling of SDN

Though the SDN provides the tremendous advantages over traditional networking, but wide spread use of it has some limitations.

Flow setup overhead:

The switches have limited computing capacity and limited bandwidth between data and control planes. Such issues can limit the rate of flow setups, even for the best implementations the controller can set few hundred of flows per second. The flow setup rate is insufficient for the high-performance network. The switches and controller are connected with fast physical medium, even in such cases the flow set up per second is limited. Flow table misses, the data plane must tweak the switches control plane to forward the packet to the controller. But the management of CPU in most of the switches are wimpy and not designed to handle per flow operations. The control plane bandwidth in switches are limited because of the cost consideration. The data plane with ASIC line card is very fast such that the switch can make forwarding decision at line rate. But the control data plane between ASIC and CPU is not frequently utilized in the traditional switch operations, therefore this is typically the slow path. A switch's limited internal bandwidth and wimpy CPU limits the data rate between the switch and the central controller. The researcher measured the latency imposed they have observed that to forward the packet ASIC takes around 5us, but the round trip time between ASIC and management CPU and the RTT is 2ms between the OpenFlow controller and CPU. The new flow is depayed by at list 2 RTTs

This flow-setup latency is far too high for high performance networks, where most flows carry few bytes and latency is critical. As a result, adding even 1ms delay to a latency-sensitive flow is "intolerable". Some researchers have observed that the delay between arrival of a TCP flow's first packet and the controller's installation of new flow-table entries can create many out-of-order packets, leading to a collapse of the flow's initial throughput, especially if the switch-to-controller RTT is larger than that seen by the flow's packets.

Increase in flow setup delay due to statistics gathering:

Flow setup and statistics gather compete for the same limited switch controller bandwidth. Higher the rate of statistics gather lower the flows the switch can setup.

Switch state size:

A limited number of flow entries can be supported in hardware. The 5406zl switch hardware can support about 1500 OpenFlow rules, whereas the switch can support up to 64000 forwarding entries for standard Ethernet switching. A single reason for such high disparity is that OpenFlow rules are stored in a TCAM, necessary to support OpenFlow's wildcarding mechanism. The TCAM entries are an expensive resource, whereas Ethernet forwarding uses a simple hash lookup in a standard memory. As the TCAM consumes lot of ASIC space and power,

it is hard to increase the TCAM memory. An OpenFlow rule is described by 10 header fields, which total 288 bits, whereas an Ethernet forwarding descriptor is 60 bits.

Controller overhead:

Controller instance can support only a limited number of flow setups per second therefore involving the controller in all flows creates a potential scalability problem. NOX controller can handle "at least 30K new flow installs per second while maintaining a sub-10 ms flow install time. The controller's CPU is the bottleneck, found that 100K flows arrive every second on a 1500-server cluster, implying a need for multiple OpenFlow controllers.

Global View:

By taking the advantage of SDN and to obtain the globally optimized state, requires the global knowledge of network traffic. The current SDN architecture is designed for the single network. To obtain the global solution in between the ASes it is necessary to share the traffic conditions in between the ASes.

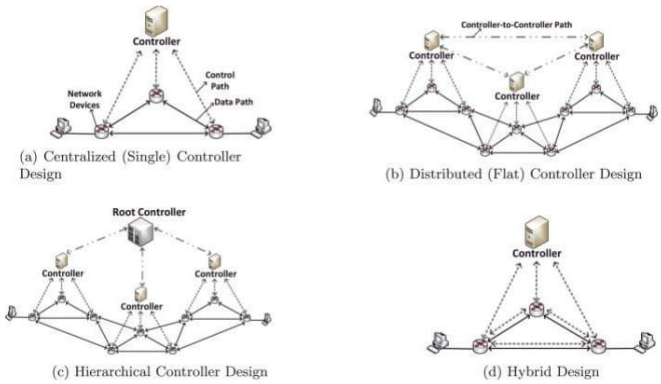
Flow setup delay:

The procedure to transfer the packet to controller, controller process and transfer the policies back to the switch takes finite time to setup the policy and establish the flow.

Security:

The flow between the controller and switch is not secured. Attacks such as man in middle , DOS can hinder the performance of SDN architecture. Even lead to malicious server, as the attacker can hijack the link between the switches and controller and install rules in switches that leads the users to malicious server.

IV. SDN ARCHITECTURES AND PROPOSED SOLUTION TO OVERCOME SCALING ISSUES



(a) Centralized (Single) Controller Design

(b) Distributed (Flat) Controller Design

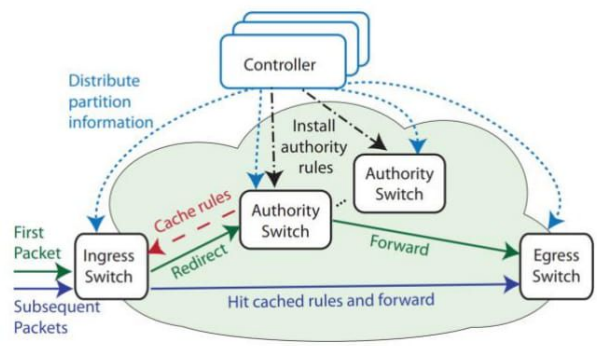
(c) Hierarchical Controller Design

(d) Hybrid Design

To overcome the limitations of SDN architectures many architectures are proposed. Different architectures will be explained in the proposed solutions.

Proposed solutions:

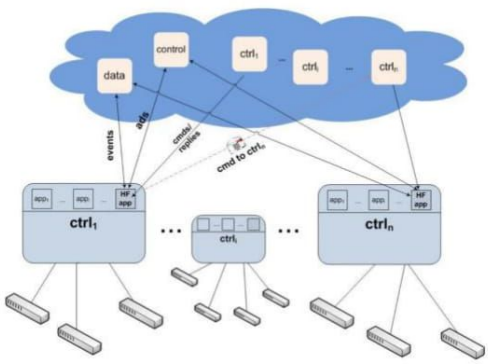
DIFANE:



The DIFANE deploys the authority switches with the commodity switches. The `authority switches install the pre-dedicated rules of controllers. The controller partitions the policies into sub policies that to be installed in the authority switches. When the first packet of the flow comes into the commodity switches, due to the cache miss the packet is redirected to the respective authority switches. The authority switch matches the packet with the respective policy, process that packet with that policy and forwards to the respective commodity switches, also installs the policy into the commodity switch. The DIFANE uses centralized SDN architecture with some modified switches. The DIFANE aim to decrease the flow setup delay by not forwarding the new packet to

the controller but to the switches which reduces the communication and processing latency. This approach also reduces the traffic between the controller and the switches and reduces the resource utilization of the controller. Disadvantage of such approach are that it does not provide global view of the traffic condition, it does not provide any secure communication between the controller and the switches and this approach need to change some switches.

HyperFlow:

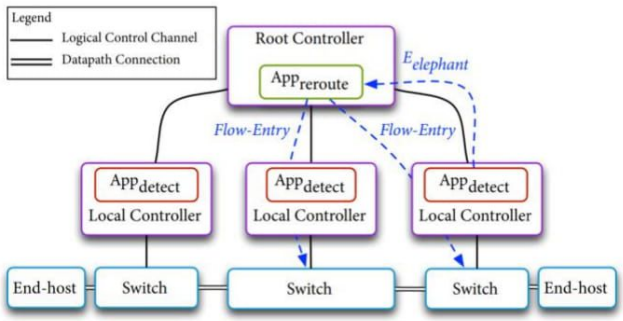


HyperFlow works on the distributed SDN architecture, it deploys controller for each autonomous system. All the controllers have a consistent network-wide view and run as if they are controlling the whole network. The controllers run the exact same controller software and set of applications. The controllers are connected with each other via link to share their traffic conditions to obtain the global view, this sharing of traffic condition does not affect working of the controllers. The HyperFlow aim to obtain the global view of the networks, it reduces burden on the controller by assigning the individual controller to the autonomous systems. Disadvantage of such approach are that the approach still have high initial setup delay because of the distance between the switches and the controllers, the approach does not provide any secure communication between the controller and switches.

Devoflow:

This approach works on centralized architecture. The approach works on partially decoupling the control plane, it configures the less important flow inside the data plane to reduce the controller traffic. The approach aims to reduce the control plane traffic between the switches and controller.

Kandoo:



The Kandoo works on hierarchical SDN architecture. It deploys the local controllers and Root controller on top of them. The local controllers are deployed for the individual networks. It takes care of local network decisions. The traffic condition of each network or statistic information of each network is give to root controller. The root controller processes the static information and obtains the global solutions. Disadvantages of this approach are that it has high initial setup delay, the approach does not provide secure communication between the controllers and switches.

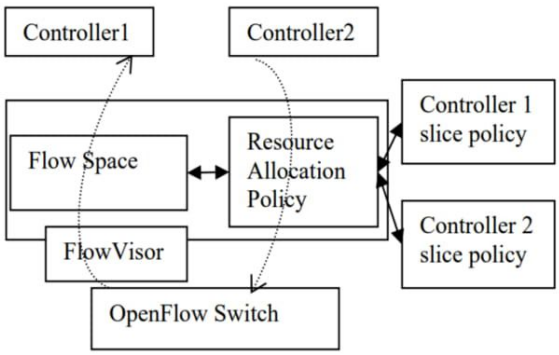
nvTCAM and sTCAM hybrid architecture:

The hybrid TCAM architecture which consists of NVM-based TCAM (nvTCAM) and SRAM-based TCAM (sTCAM), utilizing nvTCAM to cache the most popular rules to improve cache-hit ratio while relying on a very small-size sTCAM to handle cache miss traffic to effectively decrease update latency. Considering the special rule dependency, efficient Rule Migration Replacement (RMR) policy to make full utilization of both nvTCAM and sTCAM to obtain better performance.

Stats server:

This approach works on centralized architecture. The approach maintains different stats server along with the controller and uses different links to communicate with switches. Due to the partition of stats pulling and flow setup, the flow setup delay due to stats pulling decreases. To obtain the global view the stats server share information with each other.

FlowVisor:

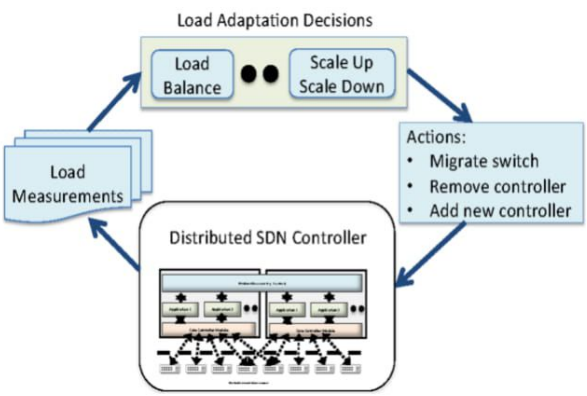


This approach uses Special OpenFlow controller depict as a proxy between switch and controller, slices the network and designates switches to appropriate controller. It also performs the vulnerability analysis. Disadvantage of this approach is that it does not provide global view.

R-SDN:

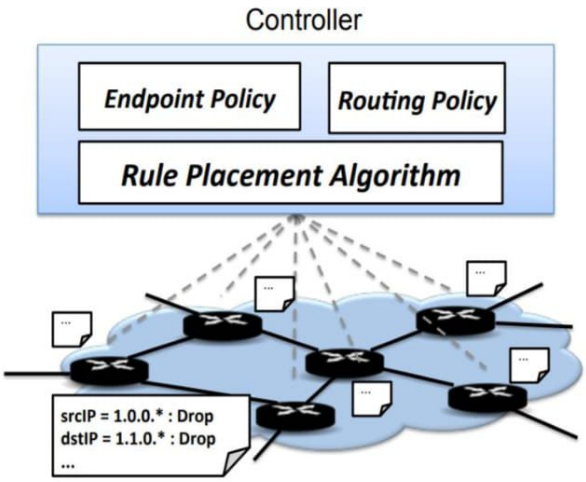
A recursive way to construct a new architecture of SDN so that the problem of scaling can be solved. A recursive algorithm is put forward to implement this new idea. As a kind of recursive model, Fibonacci series can be applied to R-SDN.

ElastiCon:



This approach scale of the number of controller based on amount of input traffic. It aims to reduce the controller resource utilization by increasing the number of controller count by increasing the traffic and decrease the count by decrease in the traffic. Disadvantages of this approach are that it has high initial setup delay, it does not provide the global view, it does not provide secure communication between controller and switches and the load balancing takes time that add latency to flow setup.

One big Switch:



This approach stores the policies along the path rather storing at starting of the switch. This approach aims to reduce the utilization of single switch resources. The disadvantage of this approach is that the packet enters into the network which should dropped at starting of the network.

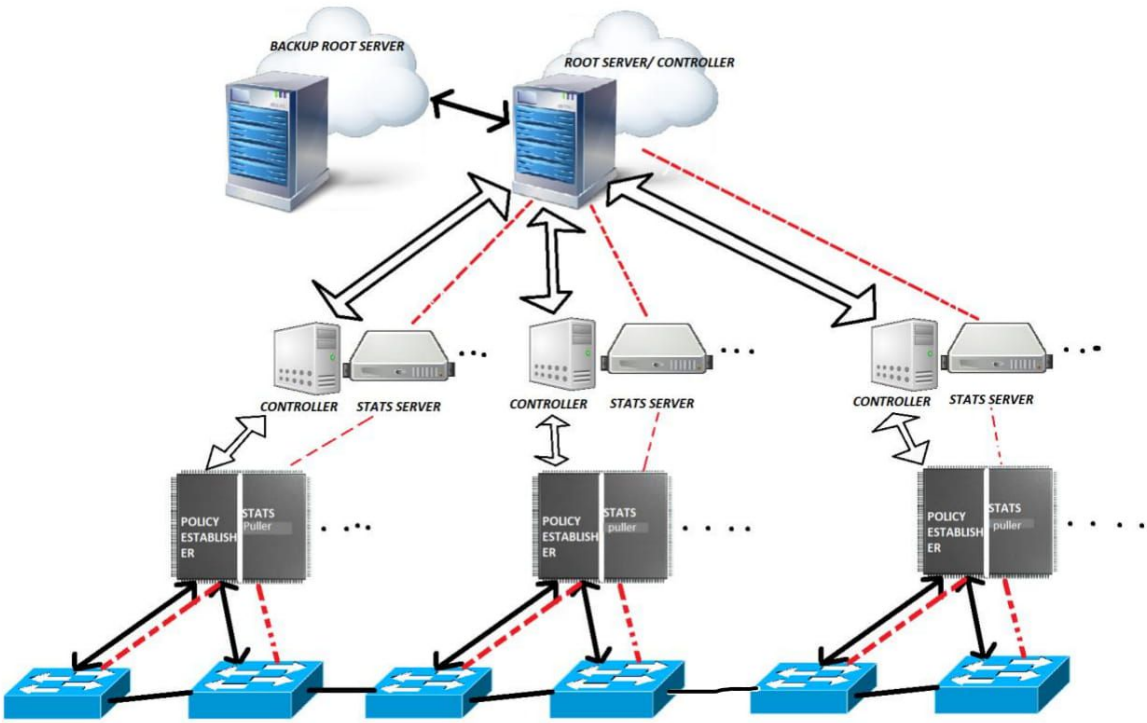
CPU as co-processing unit:

This approach keeps the traffic into data plane by configuring complete forwarding table in software of the switch. The TCAM only holds partial forwarding table for the fast flow. The approach aims to lower the traffic between the controller and the switches by storing the table in software. The disadvantage of this approach is that all the switches in this approach need to be replaced.

We have see maximum of the proposed solution but we don't find any comprehensive solution that mitigate all the problems in scaling of SDN.

In cloud computing similar problem occurred while scaling. The designer need to deal with the delay and resources available at cloud. To mitigate such issues designer used tier based approach to reduce the delay and provide the scalable resources. Such tier-based approach can be used to solve the SDN scaling problems.

V. Proposed Approach



Configuration:

The architecture consists of commodity switches. Connected with each other. Instead of connecting the switches directly to the controller. The switches are connected to embedded system. This embedded system is placed closer to the switches such that the initial flow establishment latency will be minimized by keeping the policies proximate to the switches. The embedded system is divided into two parts. The policy establisher is responsible processing the incoming packet and configuring the policies/rules into the switches. The Stats puller pull the traffic condition from the switches to obtain the statistics. By dividing the embedded system into two parts the latency in the flow establishment occurs due to resources consumed by the Stats pulling will be mitigated. The flow establishment will have different processor and have different link with the switch to the embedded system this reduces the resource consumption. The embedded system stores and transfer the polices to the controller by using hash function. It stores the hash value of the policy and transfer the hash value of the flow to the controller. While transferring the policies at the switch it converts back the original message and transfer to the switch. At the controller two separate servers are maintained. One is used for the policy configuration called as normal controller the other server pulls the continues statistics from the embedded system and stores the traffic condition. The controller is connected to the policy establisher of the embedded system via high speed optical fiber link. The stats server is connected to the stats puller via high speed optical fiber link. The two specialized server relieves each other's task. Such approach further reduces the controller resource utilization. The controller stores the hashed value of policy and transfer it to the embedded system. The local controller and Stats servers are connected to root server which gathers the stats from different controller and stats server and provides global solution to the networks. The redundant root server is maintained to avoid the single point of failure. The embedded systems and the controller can be scaled according with the incoming traffic.

Operation:
When the first new packet comes into the switch, it gets forwarded to the embedded system, embedded system processes the packet and configures the policy into the switches. If the policy for new packet is not found in the embedded system (ES) them then the ES forwards the packet to the controller. The root server fetches the stats and implement dynamic policies such as load balancing. The root server fetches the stats and implement dynamic policies such as load balancing. The policies stored in the controller and transferred to the ES/Root with HASH function.
Advantages of such approach:
The approach provides Global view.
The approach enables scaling with Backward compatibility.
The approach provides secure communication between the switches and controller.
The approach mitigates Initial setup Delay.
The approach lowers Controller involvement and utilization of the links.
Because of the separation Frequent statistics-pulling does not affect the flow setup rate.

In this paper we are going to analyze the performance of embedded systems over the normal controllers.

Experiment Setup
For the experiment we use Surface pro 4, with Intel I5 processor and 4GB RAM. We used the simulation on Ubantu Linux platform. To simulate the performance of the processors we used Gem5 simulator with Splash2 benchmark and McPAT tool.
Gem5:
gem5 is a modular discrete event driven computer system simulator platform. That means that:
gem5's components can be rearranged, parameterized, extended or replaced easily to suit your needs.
It simulates the passing of time as a series of discrete events.
Its intended use is to simulate one or more computer systems in various ways.
It's more than just a simulator; it's a simulator platform that lets you use as many of its premade components as you want to build up your own simulation system.

gem5 is written primarily in C++ and python and most components are provided under a BSD style license. It can simulate a complete system with devices and an operating system in **full system mode (FS mode)**, or user space only programs where system services are provided directly by the simulator in **syscall emulation mode (SE mode)**. There are varying levels of support for executing Alpha, ARM, MIPS, Power, SPARC, and 64 bit x86 binaries on CPU models including two simple single CPI models, an out of order model, and an in order pipelined model. A memory system can be flexibly built out of caches and crossbars. Recently the Ruby simulator has been integrated with gem5 to provide even more flexible memory system modeling.

There are many components and features not mentioned here, but from just this partial list it should be obvious that gem5 is a sophisticated and capable simulation platform. Even with all gem5 can do today, active development continues through the support of individuals and some companies, and new features are added, and existing features improved on a regular basis.

The gem5 simulator supports four different CPU models: AtomicSimple, TimingSimple, In-Order, and O3. AtomicSimple and TimingSimple are non-pipelined CPU models that attempt to fetch, decode, execute and commit a single instruction on every cycle. The AtomicSimple CPU is a minimal, single IPC CPU which completes all memory accesses immediately. This low overhead makes AtomicSimple a good choice for simulation tasks such as fast-forwarding. Correspondingly, the TimingSimple CPU also only allows one outstanding memory request at a time, but the CPU does model the timing of memory accesses.
Splash 2:
The SPLASH-2 suite is one of the most widely used collections of multithreaded workloads [16]. It is composed of eleven workloads, three of which come in two implementations that feature different

optimizations. We provide an overview in Table 1. When SPLASH-2 was released at the beginning of the 90s, parallel machines were still a relatively uncommon and expensive branch of computers. Most of them were owned by well-funded government and research institutions where they were primarily used to work on scientific problems. The composition of the SPLASH-2 suite reflects that. The majority of workloads belong to the High-Performance Computing domain.

McPAT:

McPAT is the first integrated power, area, and timing modeling framework for multithreaded and multicore/many-core processors. It is designed to work with a variety of performance simulators (and thermal simulators, etc.) over many technology generations. McPAT allows the user to specify low-level configuration details. It also provides default values when the user chooses to only specify high-level architectural parameters.

I kept the baseline 2 GHz x86 processor subsystem running the RADIX benchmark, with an L1 data cache size of 4 kB and associativity of 2, L1 instruction cache size of 4 kB and associativity of 2, and a unified L2 cache size of 128 kB with associativity of 4. I ran the simulation for 5 million instructions. To run the simulation, I used following script:

```
> build/X86/gem5.opt configs/example/se.py -I 5000000 --cpu-type=DerivO3CPU --
sys-clock=2GHz --cpu-clock=2GHz --caches --l2cache --l1d_size=4kB --
l1i_size=4kB --l2_size=128kB --l1d_assoc=2 --l1i_assoc=2 --l2_assoc=4 --
cmd=splash2/codes/kernels/radix/RADIX
```

To compare simulate ASIP ARM A9 processor I used following command.

```
> build/ARM/gem5.opt configs/example/se.py -o "-m8 -o" -I 10000000 --cpu-
type=DerivO3CPU --sys-clock=2GHz --cpu-clock=2GHz --caches --l1d_size=16kB --
l1i_size=16kB --l1d_assoc=2 --l1i_assoc=2 --cmd=splash2-
orig/codes/kernels/fft/FFT
```

X86 processor:

The x86 architecture is an instruction set architecture (ISA) series for computer processors. Developed by Intel Corporation, x86 architecture defines how a processor handles and executes different instructions passed from the operating system (OS) and software programs.

ARM A9 processor:

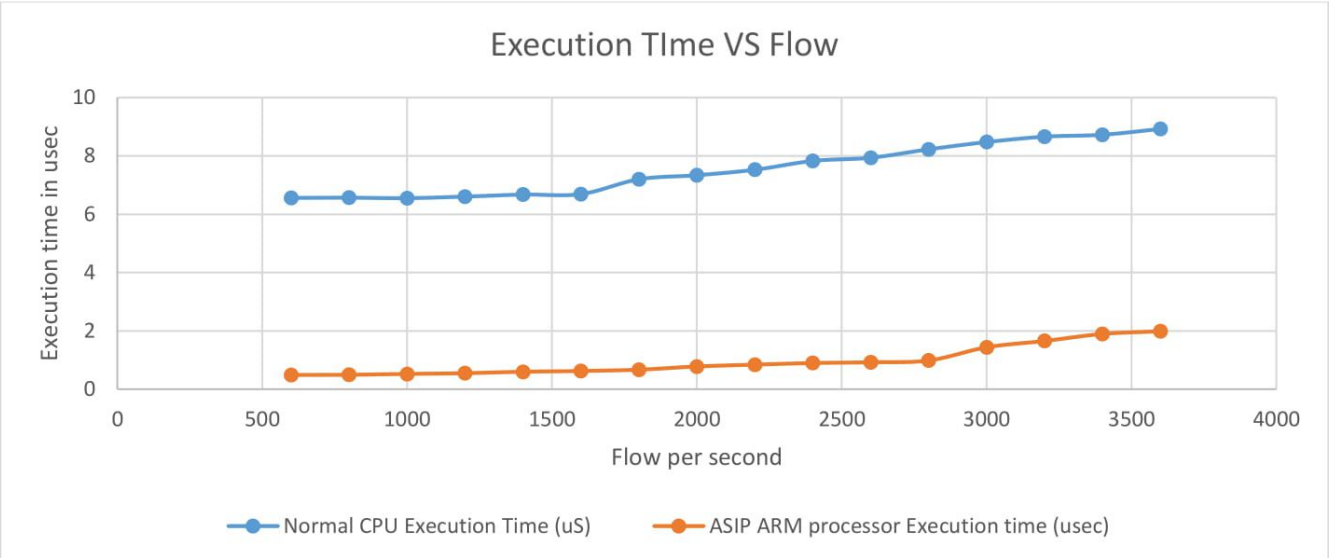
The Cortex-A9 MPCore multicore processor integrates the proven and highly successful ARM MPCore technology along with further enhancements to simplify and broaden the adoption of multicore solutions. The Cortex- A9 MPCore provides the ability to extend peak performance to unprecedented levels while also supporting design flexibility and new features to further reduce and control the power consumption at the processor and system level.

Targeted implementations of the Cortex-A9 MPCore can also offer mobile devices increased peak performance over today's solutions by utilizing the design flexibility and advanced power management techniques offered by the ARM MPCore technology to maintain operation within the tight mobile power budgets.

Results:

SR No.	Number of flows per sec	Normal CPU Execution Time (uS)	ASIP ARM processor Execution time (usec)
1	600	6.558869	0.488514
2	800	6.5658	0.49523
3	1000	6.550433	0.52387
4	1200	6.60433	0.549832
5	1400	6.67281	0.5982478
6	1600	6.690281	0.624978
7	1800	7.197842	0.6698024
8	2000	7.33654	0.778924
9	2200	7.524389	0.84267
10	2400	7.826454	0.897624
11	2600	7.937268	0.9248236
12	2800	8.224784	0.9846725
13	3000	8.473829	1.437628
14	3200	8.657438	1.6548
15	3400	8.72436	1.89423
16	3600	8.923488	1.98724

As we can see the difference in execution speed. The ASIC provide 13.42X speed that the speed of the conventional X86 processor used in controller. The controller latencies increase further with the



controller application such as floodlight. Also, the proposed approach uses optical fiber link which reduces the latencies further. The increase in the flow the execution time of conventional processor gradually where as the ASIP ARM A9 processor do no increase gradually with the flow, when it reaches to threshold it starts increasing gradually.

The McPAT simulation shows that general purpose processor takes more area and power as shown below:

	X86 Area (mm^2)	ARM A9 Area (mm^2	X 86 CUP Total power dessipation (W	ARM A9 Total power dessipation (W)
Baseline	134.729	55.0782	88.2196	37.48038
Bestcase	157.655	67.8501	91.4147	37.95148

VI. Conclusion

SDN proposes great control plane improvement over the traditional network. But due to gathering of initial packets the SDN many scaling issues. To overcome from such scaling issues many solutions where proposed, but maximum of them lack in at least one of the scaling issues. There is no comprehensive solution that mitigates the scaling issues. The paper proposes the novel architecture to overcome from such issues. This paper is focused on the embedded systems in the architectures. The simulation result clearly indicates that the ASIP processor provides great improvement over conventional controller processor in terms of latencies, power and area. Also, the embedded systems are placed closer to the switch and connected to the via high speed optical link with reduces the latencies further. The paper only measures the latencies of conventional x86 processor used as controller with the software implementation such as NOX / floodlight the latencies will increase further. In conclusion, the SDN provides great functionality in network control plane, right SDN scaling solution must be implemented to SDN without any drawback.

References

[1] Advance Networking Slides by Lorenzo de Charli
[2] A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN), MuratKarakus ArjanDurres
[3] DevoFlow: Scaling Flow Management for High-Performance Networks
[4] SwitchReduce: Reducing switch state and controller involvement in OpenFlow networks, Aakash S. Iyer; Vijay Mann; Naga Rohit Samineni
[5] On Scaling Software-Defined Networking in Wide-Area Networks, Shuhao Liu and Baochun Li
[6] Software-Defined Networking: Challenges and research opportunities for Future Internet, Akram Hakiri
[7] Unified nvTCAM and sTCAM architecture for improving packet matching performance, Xianzhong Ding
[8] Green DataPath for TCAM-based Software-Defined Networks, Huawei Huang
[9] FlowVisor: A Network Virtualization Layer, Rob Sherwood
[10] W.-K. Chen, *Linear Networks and Systems*. Belmont, CA, USA: Wadsworth, 1993, pp. 123–135.
[11] Github