International Journal of Pure and Applied Mathematics
Volume 114 No. 12 2017, 29-39
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

Enhancement of Flow in SDN by Embedded Cache

[1]K.B. Anju, Jeethu Reju[2] and [3]Leena Vishnu Namboothiri

[1]Department of Computer Science & IT,

School of Arts & Sciences, Amrita University, Kochi.

[2]Department of Computer Science & IT,

School of Arts & Sciences, Amrita University, Kochi.

[3]Department of Computer Science & IT,

School of Arts & Sciences, Amrita University, Kochi.

Abstract

The SDN has rebuilt the networking architecture to make the network control directly programmable. Decoupling the routing and forwarding functions has enabled agility and a centralized control over the network. A prolonged message request is received by the controller when a flow miss is identified in the flow table. Idle point of the SDN switch for flow rule miss may cause delays in packet forwarding and thus leads to performance degradation. By updating the flow table faster such message requests can be abolished and the controller can be released from subsequent workloads. A cache module has been introduced in to the switch in order to update the flow table faster. For a flow miss, switch checks for a cache entry and the flow table is updated based on the cache. The procedure is proposed to enhance the data flow in SDN by minimal communication with the controller.

Key Words: Cache, control plane, data plane, OpenFlow, SDN controller.

1. Introduction

An approach to SDN has enabled the network architecture in modifying the communication structure in to two, namely data plane and control plane. By dispatching the network control layer and data forwarding layer. It has helped the network control to become directly programmable and also enabled the underlying architecture to be abstracted from applications and other available services of the network [1].Every SDN model has a centralized control of the network by separating the control logic to some off-device hardware component of the computer resources. The SDN controller is said to be the "brain" of the network. It visualizes a centralized view of the whole network by allowing the network administrator the right to manage the network traffic in the underlying systems (like switches and routers). SDN uses southbound API's between the control layer and infrastructure layer to relay information between the infrastructure layer and control layer. OpenFlow, is considered as the most commonly used standard protocol for the information flow. To communicate with the application and business logic SDN use northbound API's between the application layer and control layer, which has allowed the network administrators to control the traffic through programs and to launch other network services.

The decoupled control layer and data layer in the network promise SDN a high service provisioning and agile network management. A logically centralized controller in SDN offers a global view of the network and provides a variety of network policies through the application programming interface. Controller has the power to exploit the entire knowledge of the network to optimize flow management and support other scalable and flexible services [4]. The prominent role of controller is to manage the data flow in the data plane every such flow in the network is permitted by the controller on the basis of some predefined network policies. The controller allocates a route for the flow to take if the flow policies are satisfied, each entry is added for that flow in the switches along the path associated by the controller.

The switch simply manages flow tables whose entries are populated by the controller only. Standardized protocol namely OpenFlow specifications and API's are used for communication between the controller and the switches. Initially, DevoFlow modified the OpenFlow model to minimize the overheads involved [14].The switch forward incoming packets out to the appropriate output port based on the flow table policies. The flow table includes priority information as provided by the controller. It can now drop a packet for a particular flow, temporarily or permanently based on the policies provided by the controller. The controller periodically updates the underlying switches using the OpenFlow protocol. While there is no matching entry in the flow table for the newly arrived packet, the switch immediately sends a request to the controller. The time required for this communication can lead to some consequences. It has also caused communication vulnerabilities when a flow

rule is deleted by the controller and is not updated in the switch flow table as soon as the deletion is done by the controller. The switches need to wait until a response is send by the controller at these circumstances.

Case 1: The switch will alert the controller with a message request when there is no matching entry for the arrived packet in the flow table. This will take a few Gbps to receive the request, process it and then to transfer the rule entry to the data plane.

Case 2: There are chances for security concern issues to take place when a rule entry is not immediately deleted in the flow table as soon as it is being removed from the control plane.

Main contributions of our work are summarized as follows,

- Switch performance upgraded
- Faster forwarding of data packets
- Reduction of packet transmission between switch and controller
- Enhanced network flow

2. Related Work

Ethane network architecture led to the concept of OpenFlow [9]. Though SDN emerged in 2008, it has a lot of its features of earlier technologies. The open flow protocol which is used for transmission of data packets in SDN act as a shielded interface between controller and network devices. This API is suitable for both new rules of SDN controller and flow table updating of the hardware switch. SDN architecture which used open flow is regarded as the most suitable architecture till date. Its because it provides protected transfer of control from control layer to data layer. This protocol programmatically updates the flow table entries. Very soon in 2009, SDN technologies became popular and were widely accepted. The OpenFlow protocol behaves like an API that configures switches in SDN. But a programmable interface is used for control automations in SDN [8]. DIFANE is a published work that proposes architecture which updates flow rules based on the priority [12].

The paper by MaciejKuzniar, Peter Peresini, DejanKostic about SDN flow table emphasizes that there is a constant delay between the control plane's rule installation and the data plane's ability to forward packets according to the new rule [3]. Infinite CacheFlow in Software- Defined Networks by Naga Katta, Jennifer Rexford, and David Walker proposed a caching system which implements the abstraction of an infinite switch for software- defined networks. Cache master module receives OpenFlow commands from the controller and uses the OpenFlow protocol to distribute rules to the underlying switches [5]. Network testing, an overview of approaches, by SuchitaNavpute, Sanjeev J. Wagh points out certain network problems like packet drop before it reaches the destination and packet drop due to buffer Overflow[6]. A related study to resolve the rule dependency problem that has small storage overhead has

proposed an architecture called CAB that cache the wild card rules [10]. CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks by Naga Katta, OmidAlipourfard, Jennifer Rexford, David Walker introduced a hybrid rule splicing module to provide large flow tables at low cost. CacheFlow doesn't cache individual rules [7]. But the above works doesn't help to update the flow table faster. Moreover, the configuration changes in the network that required a consistent flow table updates and controller performance remains to be an open research problem [13][15].The review has summarized to figure out some of the flow table maintenance issues in the network. The delay problem associated with unknown rule installation was identified and solutions are proposed [2].

3. Proposed System

The goal of a cache module is to enable efficient and fast flow management of update of rule. It can be either a virtual cache or a hardware cache [5].Controller periodically updates the cache with the copies of the modified rule entries which is represented in the form of a matrix in the cache [11]. Instead of sending request message packets for a flow rule miss to the controller, the switch will now check for matching entries in the cache. To compare and match flow updates in a faster manner matrix representation is used. The flow table of switch is periodically updated with the new rules from the cache. In the rare case, if the cache doesn't provide the matching entry then only the switch sends request packet to the controller.

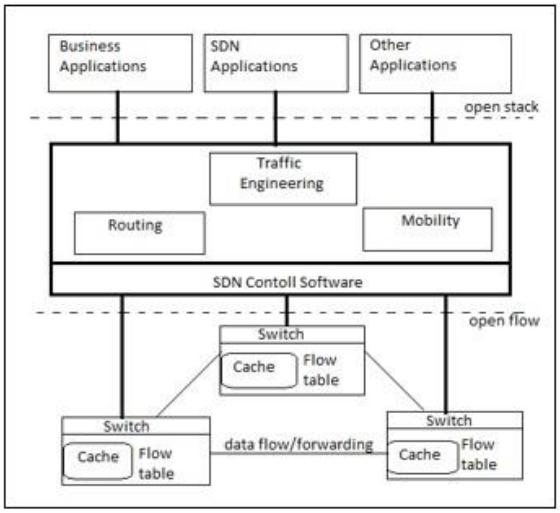


Fig. 1: SDN architecture

Deploying the cache module there will only be a one-time installation issue related with this architecture as shown in fig 1. Cache module deployed on each separate component switch gives cache more processing and I/O resources. It allows those resources to expand and contract dynamically. It has also an advantage of low latency on cache misses and not requiring an additional

hardware component [5].However, cache miss throughput is limited by the processing and I/O capacity of the local processor. Running in the hardware switch minimize the bandwidth overhead and latency for handling cache misses.

A. Cache Amendment Algorithm

The prioritized list of n rules R1, R2... Rn, where Ri is the highest priority rule than rule Rj. Every rule has a match and an action in the flow table. At regular intervals, the cache module is updated with modified rules from the controller. The controller may install rules with hard timeouts (that expire after a fixed time) or soft timeouts (that expire after a period of inactivity) [7]. For a hard timeout, the cache module sets a fixed timer; for a soft timeout, it periodically queries the traffic statistics to see if the rule has been inactive since the previous query. This has been depicted below as *cache amendment* algorithm.

for each ct

rewrite cache table based on controller rule

for each new rule R_i

insert R_i *into cache*

for each rule modified

update R_i *into cache*

for each rule deleted

delete R_i *from cache*

Cache amendment algorithm

- ct is the cache time for which the cache is periodically updated.

Every new rule that has been modified in the controller table is being updated to the cache module using the *cache amendment* algorithm. This algorithm reflects a changing circumstance that occurs frequently when every R_i is in the controller is revised or when a new rule is introduced.

The network topology below (fig 2) represents a single topology model with one OpenFlow switch and 3 hosts using the query:

>mn- -topo single,3

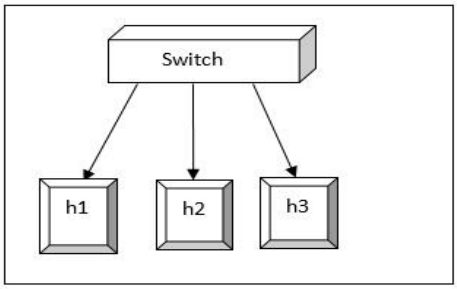


Fig. 2: Network topology

After establishing the link between the switch and host the network dump a host connection using the function dumpNodeConnections( ).

1. Topo: is a build-in base class for Mininet topologies.
2. Mininet: is the main class for creating and managing a network.
3. build(): is a function override from topology class.
4. addSwitch(),addHost(), addLink(): are build-in functions of topology class and they return switch name, host name and link key respectively.
5. start(): function to start the network.
6. pingAll(): sends packet to all nodes in the network.
7. stop(): function to stop the network
8. obj1.hosts: represent every hosts in the network.
9. dumpNodeConnections(): delete the host connection.
10. setLogLevel(info): by default output level of mininet will be set to 'info'.
11. TrailTest: Create and test a simple network.

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleTopo(Topo):
def  build(self, n=3):
      switch1 = self.addSwitch('s1')
for i in range(n):
         node = self.addHost(' i %s ' % (i+1))
self.addLink(node, switch1)

def  TrailTest():
   topo = SingleTopo(n=3)
   obj1= Mininet(topo)
   obj1.start()
print "Deleting host connections"
   dumpNodeConnections(obj1.hosts)
print "Send packets to all hosts"
   obj1.pingAll()
   obj1.stop()

if __name__=='__main__':
   setLogLevel('info')
   TrailTest()
```

Setting the output level of mininet to ‘info’, it will provide the useful informations of mininet. After obtaining the ‘info’ the controller will update cache by calling *cache amendment* algorithm.

B. Proposed Procedure for Each Data Packet Arrival

The fig 3 below is used to represent the flow of a new packet arrived at the receiving port of the switch. Let h1 represent the host that sends packet to a destination host h2. Each packet, packet_IN at the receiving port it will now check for the flow in the flow table. While the rule R_i is identified the packet is forwarded to its destination otherwise, the *flowmod* algorithm is called to update

the flow table (explained below). A message request for a flow rule miss is sent to the controller only if the missed rule is not in the cache. As a result the time delay occurred in forwarding a packet during a flow rule miss is reduced in a remarkable way. This process is repeated with every new packet arrived at the port of the switch.

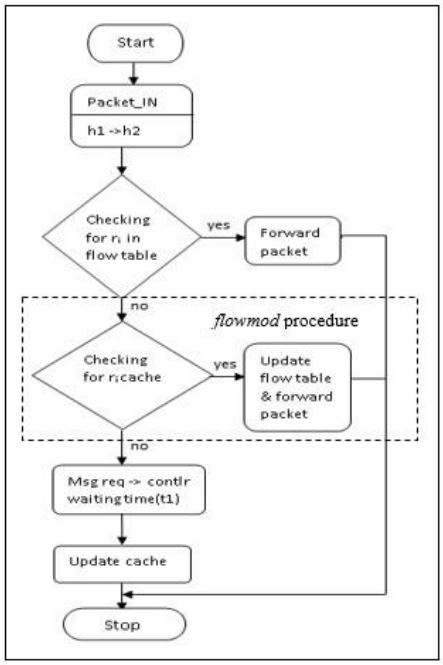
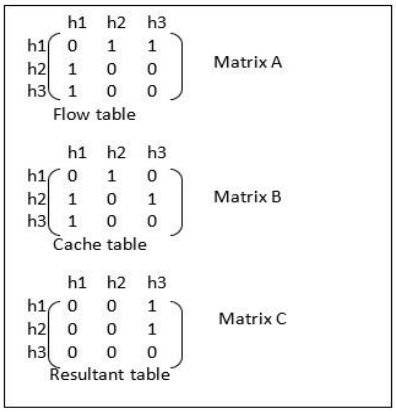


Fig. 3: Flow Chart

During a flow rule miss, *flowmod* procedure is executed and flow table is updated. For efficient comparison of flow table and cache table matrix representations are used. In matrix representation, '1' represent the presence of a flow rule between two hosts and '0' represent the absence of flow rule. The comparison procedure is implemented using a bit wise XOR evaluation of matrix.



Matrix representation

- matrix A represent flow table

- matrix B represent cache table
- matrix C represent the resultant table

flowmod Algorithm

```
for i to n do
 for j to n do
   Cij = Aij ⊕ Bij
for i to n do
 for j to n do
   if Cij = = 1 then
   {
     if Aij = = 1 then
       Aij = 0;
     else
       Aij = 1;
   }
```

flowmod algorithm

Description of flowmod Algorithm

Literally *flowmod* algorithm depicts a function which update the flow table with modified rules. Every A_{ij} from i to n where n represent the order of matrix, is compared with every B_{ij} in the matrix B. The XOR function return a new matrix C where, $C_{ij} = A_{ij} \oplus B_{ij}$

The operation XOR compares the value of A_{ij} with the value of B_{ij} and return,

$1 \leftarrow 1 \oplus 0$ and $0 \oplus 1$

$0 \leftarrow 1 \oplus 1$ and $0 \oplus 0$

C_{ij} is compared with A_{ij} and the value of A_{ij} is flipped to 0 or 1 accordingly when bit 1 is identified in matrix C.

Fig 2 represents the topology used for the proposed system. Experimentation of the procedure showed a remarkable reduction in the time delay of forwarding packets when there was a flow miss. The comparison and update of flow table with cache is effectively done using matrices. Request packet generation was done only for those entries missed in cache. Thereby processing overhead of controller for missed flow requests were reduced to an extent.

4. Conclusion

Our paper, suggests a relatively new cache module so as to update the flow table in a faster way and to make much reliable data flow in the network. In particular, our paper includes a set of algorithms and flow chart that describe the decision making during the flow miss with the aid of embedded cache. The cache module has been deployed in the hardware switch in order to reduce latency on cache misses and also doesn't require any additional hardware component. This requires on an upfront cost at the time of installation. The *cache amendment* algorithm and *flowmod* algorithm suggests a procedure to

reduce the number of unwanted request messages generated by the switch during the flow miss. Thus the performance of the controller is enhanced.

Acknowledgement

Sincere thanks to all the dignities who supported us throughout our research and fostering an academic climate. We express our gratitude to Ms. Leena Vishnu Namboothiri who has guided us all through this venture with her timely assistance, advice and constant encouragements, offered to us to make this research a success.

References

[1] Nick Feamster, Jennifer Rexford, Ellen Zegura, The Road to SDN: An Intellectual History of Programmable Networks, SIGCOMM Computer Communication Review 44(2) (2014), 87-98.

[2] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, Wu Chou, Research Challenges for Traffic Engineering in Software Defined Networks, IEEE Network 30(3) (2016), 52-58.

[3] MaciejKuizniar, Peter Peresini, DejanKostic What You Need to Know About SDN Flow Tables, Lecture Notes in Computer Science (LNCS) (2015).

[4] William Stallings, Software-Defined Networks and OpenFlow, The Internet Protocol Journal 16(1) (2013), 2-14.

[5] Naga Katta, Jennifer Rexford, David Walker, Infinite CacheFlow in Software-Defined Networks, Proceedings of the third workshop on Hot topics in software defined networking (2014), 175-180.

[6] SuchitaNavpute, Sanjeev J. Wagh Network Testing: An Overview of Approaches

[7] Naga Katta, Omid Alipourfard, Jennifer Rexford, David Walker CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks, Proceedings of the Symposium on SDN Research (2016)

[8] Garg G., Garg R., Review on architecture and security issues in SDN. International Journal of Innovative Research in Computer and Communication Engineering *2*(11) (2014), 6519-6524.

[9] Casado M., Freedman M.J., Pettit J., Luo J., McKeown N., Shenker S., Ethane: Taking control of the enterprise, ACM SIGCOMM Computer Communication Review 37(4) (2007), 1-12.

[10] Bo Yan, Yang Xu, Hongya Xing, Kang Xi, H. Jonathan Chao, CAB: A Reactive Wildcard Rule Caching System for Software-

Defined Networks, Proceedings of the third workshop on Hot topics in software defined networking (2014),163-168.

[11] PeymanKazemian, Michael Chang, HongyiZeng, George Varghese, Nick McKeown, Scott Whyte, Real Time Network Policy Checking using Header Space Analysis, NSDI (2013), 99-111.

[12] Yu M., Rexford J., Freedman M.J., Wang J., Scalable flow-based networking with DIFANE, ACM SIGCOMM (2010).

[13] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, David Walker, Abstractions for Network Update, Proceedings of the ACM SIGCOMM conference on Applications, technologies, architectures, and protocols for computer communication (2012), 323-334.

[14] Curtis A.R., Mogul J.C., Tourrilhes J., Yalagandula P., Sharma P., Banerjee S., DevoFlow: Scaling ow management for high-performance networks, ACM SIGCOMM (2011).

[15] Amin Tootoonchian, Sergey Gorbunov, YasharGanjali, Martin Casado, Rob Sherwood, On Controller Performance in Software-Defined Networks, Hot-ICE 12 (2012), 1-6.