A Framework for OpenFlow-like Policy-based Routing in Hybrid Software Defined Networks

Anshuman Mishra*, Deven Bansod†, K Haribabu‡
*Dept. of Electrical and Electronics Engineering
†‡Dept. of Computer Science and Information Systems
Birla Institute of Technology and Science, Pilani, India
{f2012074*, f2012316†, khari‡}@pilani.bits-pilani.ac.in

***Abstract*—Software Defined Networks (SDNs) provide a centralized view and allow extensive programmability of the network. They separate the control and data planes opening up immense scope in developing low cost control and management applications to operate the network. Yet, network administrators are ambivalent to revamp their entire network hardware to bring in SDN-compatible switches. This leads to the need for developing models for gradual adaptation of SDN technology. In this paper, we present a framework which allows for policy implementations based on all OpenFlow version 1.4 specified match fields, over legacy Layer 3 devices. This would enable the legacy networks to reap the benefits of SDN in an incremental, controlled and consistent manner.**

Keywords*—*Software Defined Networks, Hybrid SDN, Policy Framework, Legacy Devices, OpenFlow

I. Introduction

Software Defined Networks (SDNs) extend a centralized control of the entire network to the network administrator. This enables easy configuration of network policies and security measures in order to manage the traffic in the network. Improved visibility of the network leads to a substantial reduction in operational costs and allows for rapid and more dynamic provisioning of resources. It also offers faster recovery from link failures and makes traffic engineering much simpler. Separation of the control and data planes provides an abstraction, opening up possibilities for development of control applications to manage the network.

Despite the manifest benefits that SDN has to offer, network administrators are ambivalent towards upgrading their entire network architecture to incorporate SDN-enabled devices. The transition to SDN requires investment in new hardware along with remodeling of organisational structure to adapt to a new form of network management and application development. These impediments are the reason that the adaptation of SDN technology is meagre at best.

The need of the hour is a model which could provide SDN-like control over the network, without the need to completely revamp the entire network hardware. Vissicchio, et al. [1] have identified the research challenges and opportunities in such hybrid networks. They illustrate how SDN promises to simplify the design, operation and control of networks. At the same time they elucidate various challenges involved, such as incremental deployment, robustness and scalability that hinder full scale deployment of SDN.

In this paper, we present a framework which enables management of network traffic through the addition of some SDN-enabled switches alongside the existing legacy devices. The model provides an interface to implement SDN-like policies based on the OpenFlow [2] version 1.4 supported match fields, over the legacy devices. Each subnet within the network is required to be connected to at least one SDN compatible switch which in turn links it to the legacy Layer 3 network, though multiple subnets can connect to a single SDN-enabled switch.The SDN switch provides a gateway to monitor, control and modify the packets flowing in and out of the directly-connected subnet(s).

Even in very large networks with IP ranges analogous to class A (before CIDR), a vast pool of IP addresses remains *unused* from the perspective of the network (see Table I). Our idea aims to exploit this abundance of unused IP addresses available in the network in order to implement SDN-like policies without the need to fully upgrade the network to support SDN.

The paper is divided into the following sections: Section II reviews some existing literature relevant to the deployment of SDN and hybrid SDN networks. Section III presents the design of our model and outlines the algorithms used in the controller modules. Section IV describes the experimental setup and some policy implementation examples using the model. Section V concludes the discussion and explores the scope for future work.

II. Related Work

Several efforts have been made to facilitate the transition from legacy networks to pure SDN networks. Lu et al. proposed HybNet [3] as an abstraction layer on top of a centralized controller and non-SDN switches so that it can coordinate the communication between them. Panopticon [4] provides a software solution to determine the optimal locations for adding SDN compatible devices to achieve a network wide control in hybrid networks. ClosedFlow [5] utilizes the functionalities of proprietary devices to implement OpenFlow-like control using routers which support features like Access Control Lists (ACLs) and route-maps. The implementation is limited to devices which support the given functionalities. Further, it focuses only on emulating the basic required aspects of OpenFlow and does not provide for policy implementation based on all OpenFlow supported fields. A layer-2 solution

Telekinesis was developed by Jin, et al. [6] but it failed to provide a generalisation for a similar implementation over layer 3. In this paper, we present a framework which allows for network wide SDN-like control over Layer 3 devices and supporting policies based on all OpenFlow version 1.4 specified match fields.

III. Design

Deriving inspiration from the prior work done in this field, we propose an interface through which SDN-like control can be exercised in a hybrid SDN network. It supports policy implementations of various forms, providing extensive control and programmability of the network. The framework also allows for traffic engineering applications to work over it with a degree of delay incurred in updating the routing and flow tables.

Majority of routing protocols in legacy networks are constrained to work based on matching destination IP prefixes only. This leaves the network administrators with little flexibility to modulate network traffic based on any other criterion. To resolve this constraint, we make use of limited number of SDN compatible switches to match policy requirements. These policies are mapped onto a new set of destination IP addresses chosen from the pool of unused (free) IP addresses [See Table I for number of Free IPs available]. Simultaneously, static routes are installed in the legacy L3 devices along the path taken by the packets matching the policy using a custom driver script. The rest of the routing happens through the legacy network, which is able to route the traffic based on this new destination IP address. Finally, before reaching the destination subnet, another SDN switch maps the destination IP back to the original destination IP address of the packet. The flow entries for mapping (in the SDN switch at the source subnet) and reverse mapping (in the SDN switch at the destination subnet) are installed at the same time.

A. Model

We model the mapping of destination IPs onto the range of free/unused IPs in terms of the *limiting factors* that these mappings would bring in. A *policy path* is defined as the path which a packet would follow if it matches a certain policy. This path would be a collection of legacy routers and two SDN switches. It would start at the SDN switch directly connected to the source subnet and end at the SDN switch directly connected to the destination subnet.

Assumptions:

A.1 Topology requirements : Every subnet is connected to router(s) / L3 devices through at least 1 SDN switch.

A.2 Hybrid network before the introduction of any custom policy is functioning correctly.

A.3 The controller is aware of the details of the entire network topology.

$$\sum_{k=1}^{N} p^k = P_{total} \tag{1}$$

$$\forall k \in S, H^k_{src} + E^k_{non-map} + \sum_{i=1}^{p_k} H^i_{dest} \leq F_{max} \tag{2}$$

TABLE I: Statistics of reserved and available IPs

Statistics of reserved and available IPs		
Reserved Block address	**Address Count**	**Reference**
10.0.0.0/8	16,777,216	RFC 1918 [7]
100.64.0.0/10	4,194,304	RFC 6598 [8]
127.0.0.0/8	16,777,216	RFC 990 [9]
169.254.0.0/16	65,536	RFC 3927 [10]
172.16.0.0/12	1,048,576	RFC 1918 [7]
192.0.0.0/24	256	RFC 5736 [11]
192.0.2.0/24	256	RFC 5737 [12]
192.88.99.0/24	256	RFC 3068 [13]
192.168.0.0/16	65,536	RFC 1918 [7]
198.18.0.0/15	131,072	RFC 2544 [14]
198.51.100.0/24	256	RFC 5737 [12]
203.0.113.0/24	256	RFC 5737 [12]
224.0.0.0/40	268,435,456	RFC 5771 [15]
240.0.0.0/4	268,435,455	RFC 6890 [16]
255.255.255.255/32	1	RFC 6890 [16]
Total Reserved	**592,708,864**	–
Class A	**16,777,216**	–
Total Unavailable	**609,486,070**	–
Available for Re-map use	**3,685,481,226**	*Free IPs*

$$\forall j \in L, R^j_{def} + R^j_{IP-clashes} + p^j \leq T_{max} \tag{3}$$

$$\sum_{k=1}^{M} H^k_{dst} \leq U_{max} \tag{4}$$

where

- p^k - Number of policies at source SDN switch k
- P_{total} - Total number of Policies in the network from any source to any destination
- N- Number of new SDN switches, synonymous to number of subnets in the network as per the assumption A.1
- S - Set of SDN switches (refer to assumption A.1)
- $H_{src}{}^k$ - Number of IPs in the subnet directly connected to the kth SDN switch
- $E_{non\text{-}map}{}^k$ - Number of flow entries not used for IP mapping in the kth SDN switch
- $H_{dest}{}^i$ - Number of IPs in the destination subnet of the ith policy
- F_{max} - Maximal number of flow-entry tuples that can be added in an SDN switch
- L - Set of legacy routers/L3 devices in the core of the network, where we want to implement the policies
- $R_{def}{}^j$ - Number of routes present in the routing table of the jth router/ L3 device for legacy functioning of the network
- $R_{IP\text{-}clashes}{}^j$ - Number of routes present in the routing table of the jth router/ L3 device for mappings that handle IP clashes
- p^j - Number of policy paths through jthlegacy router
- T_{max} - Maximal number of IPv4 routes that can be added in a router in L
- M - Total number of unique paths in the network from any source to any destination
- $H_{dst}{}^k$ - Number of IPs in the destination subnet of the kth path

– U_{max} - Maximum count of Free/unused IPs available in the network

Equation 1 adds up the number of policies implemented on each subnet as source to get the total number of policies currently implemented in the entire network.

Equation 2 deals with the limits on size of flow tables in the SDN switches. It involves $H_{src}{}^{k}$ which is the number of entries required for reverse mapping the changed destination IP to the host IP at the destination SDN switch, $E_{non\text{-}map}{}^{k}$ and the summation of number of IP addresses in all the destination subnets for which unique policy paths have been defined from source S^k. The sum of these terms constitutes the total number of flows required in the SDN switches for the implementation of the policies and thus, should be less than the flow table size F_{max}. This sets a limit on the maximum number of policies which can be implemented concurrently in the network.

Similarly, **Equation 3** deals with limits imposed due to the size of routing tables in the L3 devices. It states that for every legacy router/L3 switch in the network, the sum of normal routes, routes required to resolve for Internet IP clash (refer to Section III-C) and number of policy paths passing through any given router should be less than its supported routing table size T_{max}.

Equation 4 suggests that the total IPs used for mapping, summed over all the policies implemented should be always less than the total available IPs (U_{max}).

As per the free IP address statistics presented in Table I, it is clear that even for Class A networks, the number of free IP addresses available would easily exceed the number of IP addresses required for the mapping process in a practical scenario. Moreover, the routing table size (T_{max}) is generally around 500K - 1M for a standard Cisco router [17]. Thus, even under an assumption that the legacy routes already occupy 50% of the routing tables, the size of routing tables would not pose much of a limit on the number of paths through a given router (p^j).

The number of different policy paths that an administrator can specify from a source subnet would realistically be limited only by the size of the flow table in the SDN switches. The general size of flow table is assumed to be dependent on the switch. The widely used Open vSwitch [18] supports about 500K - 1M flow entries while NoviFlow 2122 [19](an OpenFlow enabled hardware switch) supports around 125K-1M.

The equations only present constraints imposed on the total number of policies which can be concurrently implemented through the proposed framework. The terms can be viewed in a generic quantitative manner and evaluating exact values for the terms is not necessary for the implementation of the framework.

B. Controller Modules

For the implementation of the model on a network, the controller runs a few modules:

- **IP Range Allocator** :- This module manages the available IP pool, deciding which IP ranges should be allocated in order to minimize the number of IP clashes. It makes use of the network statistics from the NetStats Aggregator (discussed later) for optimal allocation.
- **Policy Translator** :- This module performs the actual mapping of a new policy to a corresponding range of IP addresses. It first verifies if the requested policy path is consistent, has all links in up state and does not contain a loop. The IP range allocated is directly mapped onto the destination subnet in order to reduce the number of flows required at the destination SDN switch.
- **Twin-flow pusher** :- After the mapping is done, this module installs flows in SDN switches at the source (for mapping) as well as the destination subnet (for reverse mapping). Along with modification of the destination IP address, the module also modifies the destination MAC address to that of the first router/L3 device in the policy path (similar to when a host sends a packet to its default gateway).
- **Legacy Route Modulator** :- The legacy devices in the network would not be aware of routing details for the new mapped IPs. This module generates static routes and installs them into the routers on the desired policy path to be implemented. The routes are configured by remote access using Telnet [20] or SSH [21]. The process can be automated using custom drivers.
- **NetStats Aggregator** :- This module periodically polls the SDN switches to get the number of matches for flows. It provides statistics to analyse patterns in the network traffic.

Algorithm 1 Twin Flow Pusher algorithm

```
procedure TWINFLOWPUSHER (S_src, S_dest, MatchConditions)
    In SDN switch connected to subnet dst,
    add entries to map IPs of map to dst.

    In SDN switch connected to subnet src,
    add entries to map IPs of dst to map,
    with appropriate MatchConditions.

    return;
end procedure
```

Algorithm 2 Legacy route Modulator

```
procedure LEGACYROUTERMODULATOR (P, M)
    P_rev = reverse of P
    for each router R ∈ P_rev do
        install static routes
        to match the prefix of subnet M,
        add entries to map IPs of dst to map,
        and appropriate next hop to next router in P_rev.
    end for

    return;
end procedure
```

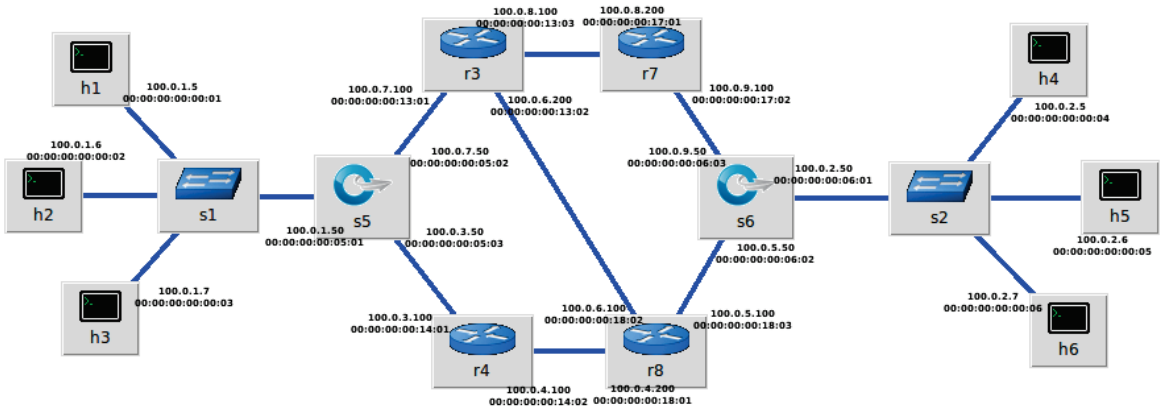

Fig. 1: Example Topology

Algorithm 3 Policy Translator algorithm

procedure POLICYTRANSLATOR (U, D, Src, Dst, MatchConditions, P)

Divide the List of free IPs U
into subnets (based on size of Dst subnet)
which can be used to map policies to.

Sort these subnets in descending order
based on usage statistics data D.

Choose the subnet which has
shown lowest Internet IP traffic.

Remove this subnet from list U.
Call this subnet as M.

call LegacyRouterModulator(P, M);
call TwinPushFlow(M, Src, Dst, MatchConditions);

return;

end procedure

The inter-relation of the controller modules is explained in the following section.

C. IP Mapping

Whenever a new policy is to be installed in the network, a set of IP addresses from the unused IP address pool is needed to map the policy onto. The policy would be stated as certain match conditions which when satisfied would route the packet through a specified policy path *P*. The Policy Translator module in the controller carries out this function of creating a one-to-one mapping of an IP range onto the destination subnet of the policy. For a destination subnet such as xxx.xxx.xxx.0/24, the module would pick up an IP address range of the form xxx.xxx.xxx.0/24 from the free IP pool and directly map the entries. This direct mapping enables us to reduce the number of flow entries required for reverse-mapping at the destination SDN switch. The controller would add a simple flow at the destination SDN switch, which matches the destination IP based on a wildcard of the form 0.0.0.255 and the ingress port and then resets the destination IP accordingly. This reduces the number of flows required for all reverse mappings to the order of number of hosts in the destination subnet.

The Policy Translator module calls the Legacy Route Modulator module before the Twin Flow Pusher module in order to retain consistency of routes in the network. The routes are added in reverse order on the path so that the source SDN switch where the destination IP is mapped, is updated at the end. Thus, the policies are implemented through a transactional interface in an *all-or-nothing* manner based on a principle similar to the controller implementation in Software Transactional Networking (STN) by Canini, et al. [22]. If any inconsistencies are encountered while setting up routes for a policy, all the changes made would be rolled back.

Each IP mapping will also require static routes to be added into the legacy routers on policy path *P* so that the legacy network is able to route the new IP range as desired. The Policy Translator would reuse IP addresses in order to map different policies which use the same policy path. This optimizes the number of static routes required in the legacy routers.

The controller can be configured to listen to SNMP traps sent out from the routers in the network in order to learn about any failures in the network. In such an event, the controller can simply perform a complete roll back by removing all the entries in the SDN switches and the legacy devices.

D. Internet IP Address Clash

The inherent problem with mapping IPs is that even these free IP addresses could possibly be associated with some host outside the entire network (For ex. a Web Server outside the network). Now, if traffic needs to be sent to such an IP address, we might end up with a clash. Our network might be configured to map a certain policy onto this IP address while unaware that this packet (intended for an external host) is not to be considered under the policy. The legacy network would thus end up routing the packet based on static routes

configured as per the policy and thus would route the packet incorrectly.

A proposed solution is to dynamically map these IPs. A packet intended to be sent outside of the network can be easily identified at the source SDN switch. A simple flow would be installed to send such packets to the controller. These flows would be installed at the same time when the flows for the conflicting policy would be installed in the SDN switch. Once the controller receives such packets, it creates a temporary map onto IP addresses from the free IP pool. Flows and static routes are added and the traffic is then routed based on this temporary mapping. Unlike policy mappings, these mappings would have an associated time to live and would be flushed if not matched for a certain time. The flushed IPs are then returned to the free IP pool.

The IP range allocator/reallocator module in the controller would analyse the traffic and optimize the allocation of IP ranges in order to avoid such IP clashes. The time-to-live of the temporary mappings can also be modulated in order to achieve a balance between delay in transmission and excessive use of IPs from the free IP pool for such temporary maps.

IV. Experimental Setup and Examples

We deploy a custom topology in mininet [23] using 6 hosts (divided as 2 subnets), 4 legacy routers R3, R4, R7 and R8. According to assumption A.1, both the subnets have a SDN switch on the path to their individual gateway router. (See Figure 1)

Components of the topology:

- Hosts - h1 to h6
  - Subnet 1 - h1 to h3 (100.0.1.0/24)
  - Subnet 2 - h4 to h6 (100.0.2.0/24)
- Legacy L2 switches - s1 and s2
- Legacy L3 switches/ routers - r3, r4, r7 and r8
- SDN-enabled switches - s5 and s6
- Controller C1 (not depicted in Figure 1) is connected out-of-band to s5 and s6

Policy Implementations:

Example 1:

All traffic with source as subnet 1, destination as subnet 2 and destination TCP port as 23 (Telnet) should follow a path P1 as s5 → r3 → r8 → s6 before reaching intended host on subnet 2

Flow entries in S5 (for mapping):

- `priority=10, ip, nw_src=100.0.1.0/24, nw_dst=100.0.2.5, tcp_dst=23 actions = mod_dl_dst=00:00:00:00:13:01, mod_nw_dst=1.0.0.5, output:1`
- `priority=10, ip, nw_src=100.0.1.0/24, nw_dst=100.0.2.6, tcp_dst=23 actions = mod_dl_dst=00:00:00:00:13:01, mod_nw_dst=1.0.0.6, output:1`
- `priority=10, ip, nw_src=100.0.1.0/24, nw_dst=100.0.2.7, tcp_dst=23 actions = mod_dl_dst=00:00:00:00:13:01, mod_nw_dst=1.0.0.7, output:1`
- `priority=11, ip, nw_src=100.0.1.0/24, nw_dst=1.0.0.0/24, actions=send_to_controller`

Flow entries in S6 (for reverse mapping):

- `priority=10, nw_dst=0.0.0.5/0.0.0.255, in_port=2 actions = mod_dl_dst=00:00:00:00:00:04, mod_nw_dst=100.0.2.5, output:1`
- `priority=10, nw_dst=0.0.0.6/0.0.0.255, in_port=2 actions = mod_dl_dst=00:00:00:00:00:05, mod_nw_dst=100.0.2.6, output:1`
- `priority=10, nw_dst=0.0.0.7/0.0.0.255, in_port=2 actions = mod_dl_dst=00:00:00:00:00:06, mod_nw_dst=100.0.2.7, output:1`

Static routes in R3:

- match 1.0.0.0/24 via 100.0.6.100 dev r3-eth2

Static routes in R8:

- match 1.0.0.0/24 via 100.0.5.50 dev r8-eth3

Example 2:

Now, if later on, administrator decides to have all TCP port 20 (FTP) traffic to follow a path P3 as s5 → r3 → r8 → s6 before reaching intended host on subnet 2.

The controller checks if this exact path is already being mapped to some IPs and finds that we are already mapping the exact same path for some other policy (Example 1), so decides not to map this policy to newer ranges but rather uses the same IP mapping without any changes required to the static routes in legacy L3 devices/routers on the way and just installs the flow-entries to map to same IPs.

Flow entries in S5 (for mapping):

- `priority=10, ip, nw_src=100.0.1.0/24, nw_dst=100.0.2.5, tcp_dst=23 actions= mod_dl_dst=00:00:00:00:13:01, mod_nw_dst=1.0.0.5, output:1`
- `priority=10, ip, nw_src=100.0.1.0/24, nw_dst=100.0.2.6, tcp_dst=23, actions= mod_dl_dst=00:00:00:00:13:01, mod_nw_dst=1.0.0.6, output:1`
- `priority=10, ip, nw_src=100.0.1.0/24, nw_dst=100.0.2.7, tcp_dst=23 actions= mod_dl_dst=00:00:00:00:13:01, mod_nw_dst=1.0.0.7, output:1`

The reverse mapping entries in S6 would not be required as we have already installed the reverse mapping entries while setting up the policy in Example 1.

V. Conclusion and Future Work

In this paper, we introduced a framework for the network administrators to implement network policies, by mapping those policies to the IP addresses which are unused inside the network. The test runs in Section IV show that we are able to implement policy-based routing in legacy routers supporting all other OpenFlow version 1.4 match fields (a total of 41), in addition to the destination IP address.

Through the tests, we have shown the examples of policies implemented through our model, though there is a need to study the impact of these mappings and reverse-mappings and interaction of controller with the legacy routers on a large network in terms of latencies introduced due to the increase in control traffic, delays due to the Internet IP clash problem and so on.

As pointed in Section III-D, the mapped IP might clash with an actual IP address in the Internet. Though we have suggested a way to handle these clashes to provide a consistency in the network, this workaround introduces its own delays. There is scope to find a way to optimize the allocation of IP addresses for mapping in order to minimize the number of clashes.

Currently, a major constraint in the IP allocation procedure is that every policy requires us to add as many flow entries in the SDN switch as there are hosts in its directly connected subnet. There is scope to implement filtering methods on the policies in order to remove redundancies within them.

Though the maximum number of policies that can be implemented in the network is realistically limited only by the flow-table size in SDN switches, use of IPv6 address space (only if the network hardware has support for IPv6) instead of IPv4 address space, would provide for a larger free IP pool availability and the idea can be explored in the future.

References

[1] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," vol. 44, no. 2, pp. 70–75. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602216

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[3] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, "Hybnet: Network manager for a hybrid network infrastructure," in *Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*. ACM, p. 6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2541602

[4] D. Levin, M. Canini, S. Schmid, F. Schaffert, A. Feldmann, and others, "Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks," in *USENIX ATC*. [Online]. Available: https://www.usenix.org/system/files/conference/atc14/atc14-paper-levin.pdf

[5] R. Hand and E. Keller, "ClosedFlow: openflow-like control over proprietary devices." ACM Press, pp. 7–12. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2620728.2620738

[6] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, and G. Jiang, "Telekinesis: controlling legacy switch routing with OpenFlow in hybrid networks." ACM Press, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2774993.2775013

[7] Y. Rekhter, R. G. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address allocation for private internets," Internet Requests for Comments, RFC Editor, BCP 5, February 1996, http://www.rfc-editor.org/rfc/rfc1918.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1918.txt

[8] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger, "Iana-reserved ipv4 prefix for shared address space," Internet Requests for Comments, RFC Editor, BCP 153, April 2012, http://www.rfc-editor.org/rfc/rfc6598.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6598.txt

[9] J. Reynolds and J. Postel, "Assigned numbers," Internet Requests for Comments, RFC Editor, RFC 990, November 1986.

[10] S. Cheshire, B. Aboba, and E. Guttman, "Dynamic configuration of ipv4 link-local addresses," Internet Requests for Comments, RFC Editor, RFC 3927, May 2005.

[11] G. Huston, M. Cotton, and L. Vegoda, "Iana ipv4 special purpose address registry," Internet Requests for Comments, RFC Editor, RFC 5736, January 2010.

[12] J. Arkko, M. Cotton, and L. Vegoda, "Ipv4 address blocks reserved for documentation," Internet Requests for Comments, RFC Editor, RFC 5737, January 2010.

[13] C. Huitema, "An anycast prefix for 6to4 relay routers," Internet Requests for Comments, RFC Editor, RFC 3068, June 2001.

[14] S. Bradner and J. McQuaid, "Benchmarking methodology for network interconnect devices," Internet Requests for Comments, RFC Editor, RFC 2544, March 1999, http://www.rfc-editor.org/rfc/rfc2544.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2544.txt

[15] M. Cotton, L. Vegoda, and D. Meyer, "Iana guidelines for ipv4 multicast address assignments," Internet Requests for Comments, RFC Editor, BCP 51, March 2010.

[16] M. Cotton, L. Vegoda, R. Bonica, and B. Haberman, "Special-purpose ip address registries," Internet Requests for Comments, RFC Editor, BCP 153, April 2013, http://www.rfc-editor.org/rfc/rfc6890.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6890.txt

[17] "CAT 6500 and 7600 Series Routers and Switches," http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/117712-problemsolution-cat6500-00.html/.

[18] "Open vSwitch," http://openvswitch.org/.

[19] "NoviSwitch," http://noviflow.com/products/noviswitch/.

[20] J. Postel and J. Reynolds, "Telnet protocol specification," Internet Requests for Comments, RFC Editor, STD 8, May 1983, http://www.rfc-editor.org/rfc/rfc854.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc854.txt

[21] T. Ylonen and C. Lonvick, "The secure shell (ssh) protocol architecture," Internet Requests for Comments, RFC Editor, RFC 4251, January 2006, http://www.rfc-editor.org/rfc/rfc4251.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4251.txt

[22] M. Canini, D. De Cicco, P. Kuznetsov, D. Levin, S. Schmid, S. Vissicchio, and others, "STN: A robust and distributed SDN control plane." [Online]. Available: https://www.usenix.org/sites/default/files/ons2014-poster-canini.pdf

[23] "Mininet," http://mininet.org/.