A Prototype of Policy Defined Wireless Access Networks

Hung X. Nguyen*, Thien Pham†, Khanh Hoang†, Duong D. Nguyen†, Eric Parsonage‡
*Teletraffic Research Centre,†School of Electrical and Electronic Engineering,‡School of Mathematical Sciences
The University of Adelaide, Adelaide, SA 5005, Australia
Email: *hung.nguyen@adelaide.edu.au, †thien.pham@oasissystems.com.au, †buikhanh.hoang@student.adelaide.edu.au,
†duong.nguyen@adelaide.edu.au, ‡eric.parsonage@adelaide.edu.au

***Abstract*—In the past few years, significant progress has been made in using software defined networking to increase automation, improve network security, simplify network configuration and reduce human effort to establish and maintain the network. There are now a vast number of studies exploring how to utilise policies to achieve the above goals. In this paper, we apply policy defined networking to wireless access network functions. We describe the details of our prototype policy defined networking solution that automatically translates high-level policies into device level implementations. We develop a novel metagraph model that can be used for policy specification, verification and refinement. We show that sophisticated traffic engineering policies can be implemented automatically on commodity hardware using our framework.**

***Index Terms*—Software Defined Networking, Policy Defined Networking, Wireless Access Networks**

I. Introduction

Over the last few years, there has been significant research into the application of software defined networking (SDN) to improve wireless access networks. These include numerous innovative approaches for access network selection [1]–[3], mobility management [4], dynamic QoS and traffic engineering [5], QoS management across wired and wireless networks [6], [7]. The ultimate goals of this research are to reduce costs, increase business agilities, and accelerate the time to market of new access network services.

Even though these solutions utilise SDN architecture, they still require manual configurations of networking devices, using either OpenFlow switching rules or traditional router/switch configurations. Furthermore, they do not have a built-in checking and debugging system for network policies. Using these solutions, network engineers need to check manually that the policies are implemented correctly and that a new policy does not conflict with existing policies on all network devices [8].

In this paper, we develop a prototype of a policy defined networking (PDN) solution for wireless access networks. Our solution automatically translates high-level policy specifications to low-level device configurations and provides built-in debugging and verification of the policies. Our contribution is twofold. First, we develop an SDN solution for programmable wireless access networks using commercial off-the-shelf hardware. We show that by using SDN we can implement advanced networking functions including firewall, DHCP, malware detection, traffic engineering on generic hardware, instead of specialised middleboxes. We explain in detail our design choices and configurations in this paper. Second, we develop a policy defined model based on metagraphs [9] to automatically check and translate high-level policies to OpenFlow statements. Our rigorous algebraic framework guarantees a correct implementation of the policies, specified using a high-level graphical user interface. We show through our prototype that these networks can greatly benefit from the rigorous mathematical policy framework provided by metagraphs. And thus, our prototype PDN solution holds great promise for enabling error-free, agile and secure network management and control for future wireless access networks.

Our solution is still in an early development state and we plan to extend the work in multiple directions. Our code and data are publicly available.

II. Background

A. Wireless Software Defined Networking

SDN was developed in an attempt to simplify networking and make it more secure. This has been achieved by separating the control plane (the controller which decides where packets are sent) from the data plane (the physical network which forwards traffic to its destination) [8]. An SDN network can be defined by three fundamental abstractions: forwarding, distribution, and specification. The forwarding abstraction (OpenFlow [10] is a popular realisation) allows forwarding behaviour desired by the network application while hiding the details of the underlying hardware. The distribution abstraction shields the SDN applications from the vagaries of distributed state and is realised by a network operating system (NOS). The specification layer is most relevant to our work on policy defined networking. This layer allows a network application to express desired network behaviour without being responsible for implementing that behaviour itself. The interface between the network application and the network operating system is called North Bound Interface (NBI). Generally, NBI is implemented using a programming language such as NetKAT [11] or REST/JSON API [12].

Separation of control and data plane exists in the wireless domain, independent of SDN. Several years ago, the IETF standardised the Control And Provisioning of Wireless Access Points (CAPWAP) [13] protocol, which centralises the control

plane in wireless networks. In principle, CAPWAP applies to all technologies although so far only the binding for 802.11 has been identified. With CAPWAP, control frames are delivered to a central controller, which is responsible for MAC layer control. A novel paradigm for re-programming of wireless interfaces was proposed in [14] where wireless nodes execute a wireless MAC processor used to program specific MAC protocols. In this architecture, the central controller can dynamically upload the protocol to use at a given point in time. Related to programmable networks is the concept of software defined radio (SDR) [2], [15]. SDR allows programmable physical layer and is a complement to the focus of this work on the network layer.

B. Policy-Based Network Management

Policy-based network management (PBNM) [16] is a well-known approach that allows top-down configuration where device configurations are derived from high-level specifications, i.e., policies. Policies are intended to capture how a user wants the network management software to behave. Through a number of policy refinement steps, these policies are translated (automatically or semi-automatically) to device configurations. PBNM presents several benefits such as less manual configurations and fewer errors, automated analysis and verification based on a formal foundation, dynamic checking, and adaptation at runtime.

A fully automated refinement of high-level policies down to low-level policies is still a topic of much ongoing research [17], [18]. In the last few years, significant evolution in SDN technologies has accelerated the development of fully automatic PBNM solutions for SDN-based networks [10]. Major recent developments in policy languages for SDN have led to a number of new languages such as NetKAT [11] and Pyretic [19].

As policy languages mature, recent researches have focused on software and applications that use these languages to enable fully automatic policy refinement. One important part of these efforts is on policy conflict resolution [17], [20]. Previous works on PBNM concentrated on policies that are applied to well-defined endpoint groups of users or resources, and often support only policies that apply directly to them without middleboxes. Furthermore, they lack the capability for analysing the policies other than simple end-to-end reachability analysis. Our metagraph based policy defined networking solution aims to address these problems.

III. System Architecture

Our prototype has two major components. The first component is the hardware that implements the OpenFlow protocol and supports various wireless access technologies such as WiFi, 3G and LTE. The second component is the software that automatically translates high-level network policies to OpenFlow statements. We describe these two components in detail in this section.

A. Hardware

We choose to use software switches running on standard CPU to implement OpenFlow and using USB network adapters to enable various wireless access technologies. More specifically, we use embedded board Raspberry Pi 2 model B+ running OpenvSwitch (OVS) as the commodity hardware to implement switching functionalities of the OpenFlow protocol. The switch is powered by an ARM v7 quad core running at 900MHz with 1 GB SDRAM. It runs on Debian Jessie 8 Operating System. OVS version 2.3.0 is installed to provide the OpenFlow switching capabilities.

The ARM board is connected to a USB hub with 4 ports. Two of these ports are connected to 2 Belkin USB WiFi dongles for WiFi connectivity both in infrastructure and ad-hoc modes. One port is connected to a 4G dongle to provide cellular access. The other port is not used presently. By using software switches and wireless adapters, we have turned a generic Raspberry Pi into a fully functional OpenFlow switch with wireless interfaces. We call this device the *RaspSwitch*. We currently have two RaspSwitches inter-connected with multiple HP OpenFlow switches in our lab. In the following sections, we provide the detailed configurations of each interface in our RaspSwitch and our SDN controller.

B. Network Interfaces Configuration

1) Wireless Configuration: There are three wireless interfaces available on RaspSwitch namely wlan0, wlan1 and wlan2. Each interface connects to a different wireless network. Interface wlan0 is connected to an enterprise WiFi network (UofA WiFi) using wpa_supplicant with full encryption. Wlan1 is connected to a commercial 4G network via a WiFi hotspot. Wlan2 is connected to an ad-hoc 802.11 wireless mesh network running B.A.T.M.A.N [21].

2) OpenvSwitch Bridge Configuration: To convert the standard USB ports into OpenvSwitch ports, we use ovs-vsctl tool in out-of-band control mode to create OpenvSwitch(OVS) bridges as shown in Figure 1.

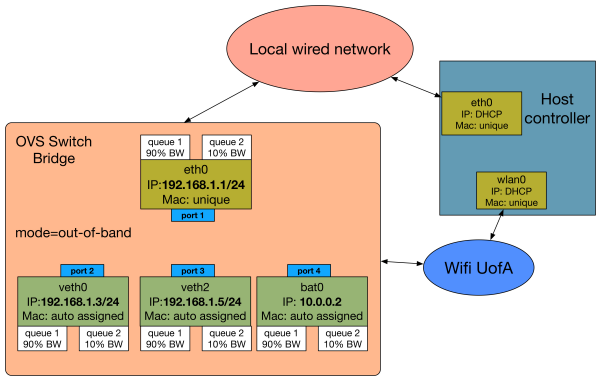

Fig. 1. OVS Bridge Configuration

The out-of-band mode enables the bridge to separate OpenFlow protocol packets from ordinary traffic packets. Using out-of-band mode, the host running controller application

for OVS bridge simultaneously connects to UofA WiFi for communicating with OVS bridge and to the local network. The interfaces, namely eth0 (physical), veth0 & veth2 (virtual) and bat0 (virtual, created by B.A.T.M.A.N protocol) are then added to the OVS bridge. Network traffic is directed to the OVS bridge via these ports to form the data plane.

3) Iptables and Routing Tables Configuration: We use *iptables* for Network Address Translation (NAT) to provide connectivity to the outside Internet via the RaspSwitch. Fig. 2 illustrates how packets are forwarded using routing tables.

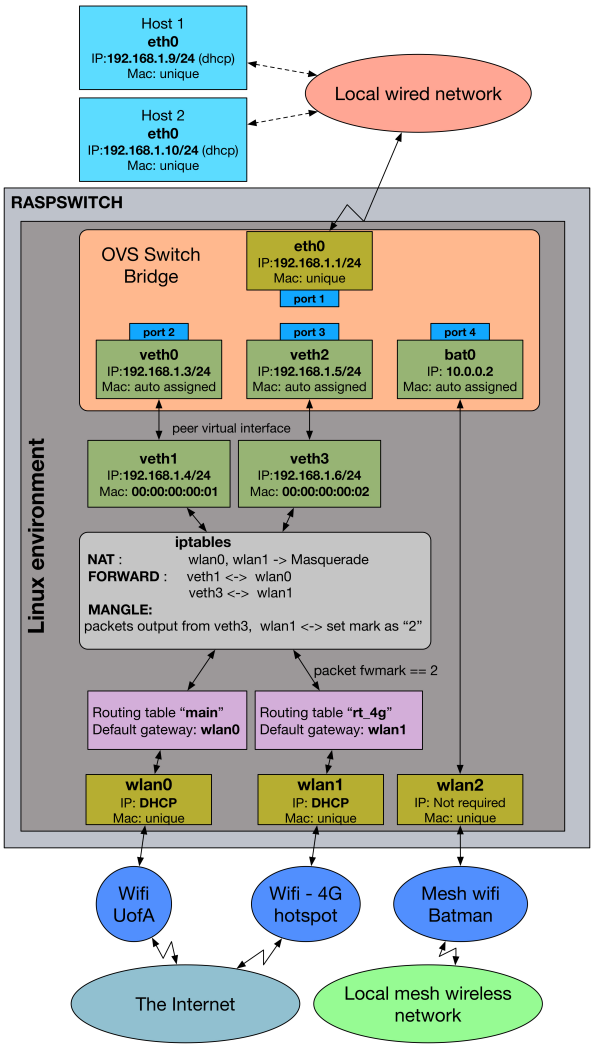

Fig. 2. Iptables and routing tables configuration.

We use standard packet masquerading techniques to turn wlan0 and wlan1 into NAT-enabled interfaces [22]. Packets are bi-directionally forwarded between interfaces veth1 and wlan0 and between veth3 and wlan1. To enable simultaneous transmission on wlan0 and wlan1, two routing tables are required for each interface respectively. In our configuration, the main routing table has wlan0 as the default gateway, and the routing table rt_4g (i.e. routing table 4G) has wlan1 as the default gateway. Every packet from veth3 and wlan1 (PREROUTING) is marked with a given ID number (2 for example). Thus if a match is found, the packet is forwarded using this PREROUTING.

C. Control Plane Implementation

We use POX [23] as our controller. POX is well-known for rapid prototyping of SDN controllers. We set up the RaspSwitch to allow multi-channel communications with the controller from other applications as shown in Figure 3.

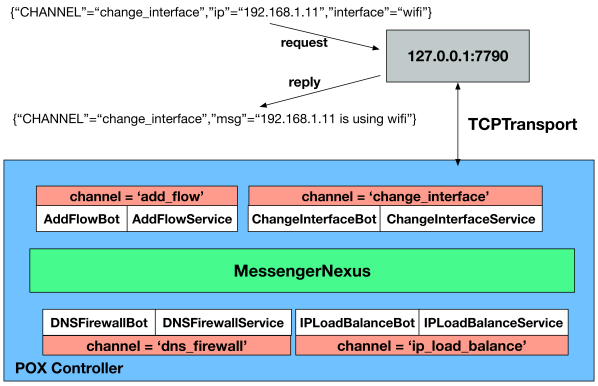

Fig. 3. North bound interface: Communication with the controller through a JSON API

Each channel has one bot (auto response) and one service (to invoke command). There are several options for the north bound interface between the applications and the controller. We use a JSON API that allows applications to transparently call controller functions. We also have developed a WEB GUI for managing the interface and routing configurations. For example, using the JSON interface, when a user wants IP traffic to and from 192.168.11 to use the WiFi interface. He can send the following statement to the controller:

$$\{CHANNEL = change\_interface, ip = 192.168.1.11,$$
$$interface = wifi\}$$

IV. Graphical Policy Defined Networking Engine

The next major component of our SDN system is a software that automatically translates high-level policies to OpenFlow statements for the RaspSwitch. For this purpose, we develop a formal algebraic model for policy specification, automated verification and deployment. Our model extends the graphical model in [17] to provide full metagraph properties [9].

A. Metagraph Model for PDN

A metagraph is a directed hypegraph, which is a collection of directed set-to-set mappings. Each set is a node in the graph and each directed edge represents the relationship between the sets. Although as simple as its definition, metagraph has powerful theoretical tools that can be used to analyse, optimise and troubleshoot policies.

Network policies are applied to endpoint groups [17]. Each endpoint group is specified by the policy designer using users, time, locations or any other criteria. To implement network

policies, a node in the metagraph contains a subset of the endpoint groups. In principle, edges on a metagraph can represent any function on the sets. Here, we use an edge to encode either a mapping function from one endpoint group to another or a filtering function allowing or disallowing one endpoint group to reach the other endpoint group. For example, an edge can represent the combination of students and time into the set of "students in class" at a given time.

Formally, we can encode the set of network policies using a meta-graph with the following structures:

$$MG = (GeneratingSet, Vertices, Edges),$$

where

- $GeneratingSet$ is the set of all users and resources;
- $Vertices$ represent the network endpoint groups;
- $Edges$ represent policy action(s) applied to the connection between two endpoint groups. The actions could be a transforming rule to map from one endpoint group to another or it could be a filtering action that allows or disallows one endpoint group to reach the other.

Alternatively, we can view the metagraph as a set of triples (Invertex, Outvertex, Attributes). Each of these triples represents a constraint of a policy. Intervex and outvertex represent the two endpoint groups and attributes are the mapping or filtering function that we want to apply to the communication between those two endpoints. A policy consists of one or several of the constraints. To build a metagraph for policies, the policy designers need to provide two inputs:

- Rules to group users and resources from the Generating Set into endpoint groups. The endpoint groups can be specified using attributes such as users, time and locations.
- A constraint on the communication between endpoint groups. For example, to allow a certain endpoint group of users to access a certain group of resources.

Once a metagraph is built, we can use well-known metagraph algorithms to analyse the policies. For example, to check if there is a policy conflict, we can check if there are paths from a node representing a user to two different nodes representing both allowed and disallowed accesses to the same resource.

B. Graphical User Interface for Policy Specification

We have developed a WEB graphical user interface for building the bipartite policy graph. The WEB GUI allows users to create and modify groups of network endpoints using drop-down menus. These groups could be a group of web servers or a group of internal network hosts. Network endpoints in these groups can be represented by their IP or MAC addresses and serve as fundamental elements in our metagraph models.

Currently we implement only policies that can be specified using "Event-Condition-Action" which can be specified via a pull-down menu. Our interface has the simple rule-based syntax of

If A and/or B then C

where A matches on some predefined sets, B is the destination and C is the action. Each of these statements is represented in a metagraph as a triple (Invertex, Outvertex, Attributes). Furthermore, our prototype only supports access control attributes such as "Allow" or "Disable" communication between two endpoint groups; and QoS attributes such as the priority level (integer numbers) for flows between two endpoints. As we will show in Section V, these restricted constructs are enough to implement very advanced traffic engineering policies.

C. Policy Verification

As the endpoint groups and the actions can be arbitrarily created and selected by the policy designer, conflicts can occur between the constraints. For example, if two groups share a user but the constraints imposed on these groups specify conflicting actions or attributes, there will be a conflict. We use metagraph and graph algorithms to detect and resolve the conflicts as follows.

Our starting point is the metagraph that was constructed in the previous section. Consider the policy in Figure 4 with three triples of (Invertex, Outvertex, Attributes). Each of these triples represents a constraint on the communication between two endpoint groups.

Constraints	Endpoint Group 1	Endpoint Group 2	A/C Action	Traffic Priority
1	D2	W1	Disabled	N/A
2	D1	W2	Allowed	Level 2
3	D3	W1	Allowed	Level 2

Fig. 4. An example policy with conflicts. There are five endpoint groups D1 = {h3, h5, h4}, D2 = {h0, h1, h2, h3}, D3 = {h2, h3, h5, h6}, W1 = {ws1, ws2}, W2 = {ws2, ws3}

The metagraph for the policy in Figure 4 is shown in Figure 5. Using this metagraph, detecting conflicts becomes a trivial exercise of reachbility analysis on metagraph. We present here an implementation of such algorithms. Many possible advanced analyses on metagraphs are provided in [9].

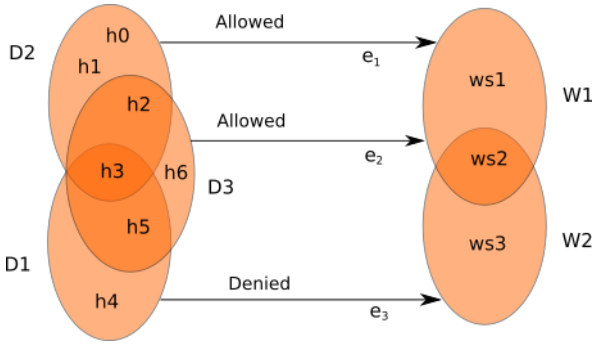

Fig. 5. The metagraph representing the policy shown in Fig. 4

The metagraph derived from the policy has five vertices D1, D2, D3, W1, W2 with three edges (D2, W1, "Access Allowed"), (D3, W1, "Access Allowed"), (D1, W2, "Access Disabled"). The generating set of this metagraph is {h1, h2, h3,

h4, h5, h6, ws1, ws2}. There are a number of conflicts in this policy. For example, there is a conflict between Constraint 1 that allows host 3 (h3) to reach web server 2 (ws2) and Constraint 2 that does not allow h3 to reach ws2. These types of conflicts occur frequently in access control policies specified on groups of endpoints [17].

In order to detect conflicts in a policy, we apply the following three steps:

- First, we normalise the original metagraph and turn it into a flat or plain graph where common parts of the metagraph vertices will be extracted and form disjoint subsets, which then serve as vertices in the normalised graph. For the metagraph in Fig. 5, the end point groups D1, D2, D3, W1, W2 are broken down into subgroups: {h0, h1}, {h2}, {h3}, {h4}, {h5}, {h6}, {ws1}, {ws2}, {ws3} as shown in Fig. 6. These are atomic elements where policy constraints are applied.
- Edges of the meta-graph then are reassigned to the disjoint atomic policy elements by mirroring the original edges between the two supersets. For example, there are two edges between h2 and ws2, inherited from the edges e1 between D2 and W1 and the edge e2 between D2 and W1. Both edges allow h2 to reach ws2.
- Once the simplified graph with disjoint endpoint groups is created, we can then apply reachability analysis between every pair of endpoint groups to detect conflicts. For example, the endpoint group h5 can reach ws2 and cannot reach ws2 at the same time due to the two original edges (D1, W2, "Denied") and (D3, W1, "Allowed").

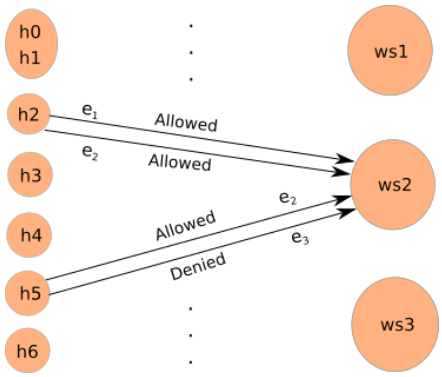

Fig. 6. The flat (plain) graph derived from the metagraph in Fig. 5. For demonstration purpose, only connections between nodes {h2}, {h5} and {ws2} are shown.

As mentioned previously, our current implementation only supports actions such as"Allow access"/"Disable access" and the traffic priority levels. Reachability analysis on a graph with these attributes can be achieved using equality comparison of the edge attributes. For more complex attributes such as a range of ports or bandwidths or a chain of services or middleboxes, this function will need to be extended to fit the intents of the policy composer. We plan to extend our metagraph model to include these functions in future work.

After the verification step, we have a list of all possible conflicts and they are reported to the user via our policy interface for final decisions.

D. Policy Switch Deployment

Once a policy passes the verification, the analysis engine will generate a set of OpenFlow statements from the conflict-free metagraph. We run a generalised shortest path algorithm on the graph to determine the next hop for each end-to-end flow between any pair of endpoint groups. Note that this algorithm is similar to the Dijkstra algorithm but works on a generalised algebra, not the $(min, +)$-algebra.

OpenFlow-equivalent statements are then derived directly from the output of the shortest path algorithm for each pair of endpoint groups. These statements will then be deployed to the OpenvSwitches via the built-in messenger service API of our POX controller. Note that we are not performing optimisation on the number of OpenFlow statements.

V. Case Studies: Wireless Access Network Applications

We describe here the applications that we have implemented in our PDN prototype in addition to the access control application presented above.

A. Access Network Selection

We use our policy defined networking testbed to implement several radio access technology (RAT) selection algorithms. These algorithms allow the users to switch automatically from one RAT to another based on performance statistics such as loss, delay, link load or fixed preferences.

Recall that our testbed currently has 2 RATs: 4G and WiFi (in infrastructure and ad-hoc modes). The policy graph is used to specify which connection is used for a given set of users to reach the Internet (networks with IP address ranges outside of the internal subnet). Once the policy is defined, the analysis engine checks for consistency and then launches a program to start collecting performance metrics that are used for switching. If a switching of RAT is triggered (from the measurement results), the controller then uses Gratuitous ARP packets to initiate the switching.

Gratuitous ARP works by sending an ARP reply (not incurring by ARP requests) to a specific host to actively invokes the host to acknowledge the MAC address associated with the intended IP address. For example, if the ARP reply contains source IP of 192.168.1.1 (gateway) and MAC address of 00:00:00:00:00:01, the host would "ack" the gateway with MAC address of veth1. Therefore, eventually the host's packets will be delivered to veth1 through the WiFi interface. Vice versa, if the host is announced that the gateway is at 00:00:00:00:00:02 (veth3's MAC). Packets from this host will pass through veth3 and to the 4G interface.

B. Traffic Prioritisation

The second application is traffic prioritisation of flows from different sources. To specify the prioritisation policy, each flow will be specified with either a high or a low priority by labeling

the edge connecting the source and destination groups with the label "high" or "low".

Once the policy is defined and the policy graph is built, it will be translated to the flow level configurations as follows. We first create two queues in each of the OVS bridge. One is given 90% of the available bandwidth and the other 10%. High priority flows will be assigned to the queue with 90% of bandwidth and low priority flows will be assigned to the low priority queue. For example from the application interface, we compose a policy that allows host 1 and host 2 to have connections allowed to a certain web server WS (IP addresses of these three endpoints are given). (h1, web_server WS) is assigned with traffic priority "high" and (h2, web_server WS) is assigned with traffic priority "low". Here we map "low" and "high" to queue 1 (90% of bandwidth) and queue 2 (10% of bandwidth) respectively.

In all of our tests, our PDN solution achieved the maximum throughput available on each of the RAT connections (WiFi or 4G).

C. DNS Malware Detection and DHCP Service

We use our policy defined networking framework to implement DNS monitoring including catching and/or blocking suspicious DNS queries, redirecting DNS queries, and altering DNS replies. A record of domains and their associated IPs are logged for other purposes.

The DNS Malware Detection is implemented by inspecting packets with a source port or destination port of 53. By checking the queries against known blacklist IPs obtained from publicly available blacklists such as Spamhaus (https://www.spamhaus.org), we can detect suspicious activities in the network. We can also isolate and quarantine the computers that generate these queries. Furthermore, by blocking outgoing DNS, the controller can stop users from accessing specific domains. We can use this capability to implement policies such as "during lectures from 10 am to 11 am, students are restricted from accessing facebook.com".

We also configure our system to provide DHCP services on the OVS bridge ports (such as eth0, bat0, etc) using POX's DHCPD class. Most of the parameters are policy-definable such as subnet, dynamic IP range, etc.

VI. Conclusion

We present in this paper a policy defined networking framework using metagraph models for wireless access networks. Our focus is on wireless functions. We show that these networks can greatly benefit from the rigorous mathematical policy framework provided by metagraphs. And thus, our prototype PDN solution holds significant potential for enabling error-free, agile and secure network management and control for future wireless access networks.

References

[1] M. Mihailescu, H. Nguyen, and M. Webb, "Enhancing wireless communications with software defined networking," in *Military Communications and Information Systems Conference 2015*, Nov 2015, pp. 1–6.

[2] V. Sagar, R. Chandramouli, and K. P. Subbalakshmi, "Software Defined Access for HetNets," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 84–89, Jan 2016.

[3] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar, "Blueprint for introducing innovation into wireless mobile networks," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, ser. VISA '10, 2010, pp. 25–32.

[4] Y. Li, H. Wang, M. Liu, B. Zhang, and H. Mao, "Software defined networking for distributed mobility management," in *Globecom Workshops (GC Wkshps), 2013 IEEE*, Dec 2013, pp. 885–889.

[5] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *Proceedings of the second workshop on Hot topics in software defined networks*, ser. HotSDN '13, 2013.

[6] M. J. Yang, S. Y. Lim, H. J. Park, and N. H. Park, "Solving the data overload: Device-to-device bearer control architecture for cellular data offloading," *Vehicular Technology Magazine, IEEE*, vol. 8, no. 1, pp. 31–39, March 2013.

[7] X. Jin, L. Erran Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and Flexible Cellular Core Network Architecture," in *Proceedings of the 9th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '13, 2013.

[8] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[9] A. Basu and R. W. Blanning, *Metagraphs and Their Applications*, 1st ed. Springer US, 2007.

[10] 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.

[11] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT: Semantic foundations for networks," *SIGPLAN Not.*, vol. 49, no. 1, pp. 113–126, Jan. 2014.

[12] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[13] P. Calhoun, M. Montemurro, and D. Stanley, "Control and Provisioning of Wireless Access Points (CAPWAP) Protocol Specification," Tech. Rep., 2009, RFC 5415.

[14] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello, "MAClets: Active MAC protocols over hard-coded devices," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT, 2012, pp. 229–240.

[15] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: A programmable wireless dataplane," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12, 2012, pp. 109–114.

[16] G. Waters, J. Wheeler, B. Moore, A. Westerinen, and L. Rafalow, "Draft IETF Policy Policy Framework Architecture," Internet Draft, Internet Engineering Task Force, Feb. 1999.

[17] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "PGA: Using graphs to express and automatically reconcile network policies," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 29–42.

[18] W. Han and C. Lei, "A survey on policy languages in network and security management," *Computer Networks*, vol. 56, no. 1, pp. 477 – 489, 2012.

[19] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with pyretic," *USENIX Login*, vol. 38, no. 5, Oct 2013.

[20] A. K. Bandara, E. C. Lupu, and A. Russo, "Using event calculus to formalise policy specification and analysis," in *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, June 2003, pp. 26–39.

[21] D. Johnson, N. Ntlatlapa, and C. Aichele, "Simple Pragmatic Approach to Mesh Routing Using BATMAN," in *IFIP Symposium on Wireless Communications and Information Tech. in Developing Countries*, 2008.

[22] M. H. Jang, "Basic Networking," in *LPIC-1 in Depth*. Nelson Education, 2009.

[23] M. McCauley, "POX," 2012. [Online]. Available: http://www.noxrepo.org/