Achieving Policy Defined Networking for Military Operations

Hung Xuan Nguyen and Michael R Webb
Centre for Defence Communications and Information Networking (CDCIN)
The University of Adelaide
hung.nguyen@adelaide.edu.au

Sanjeev Naguleswaran
QSpectral Systems Pty Ltd
Brisbane, Australia
sanjeev@qspectral.com.au

***Abstract*—There has been significant recent progress in the area of software defined networking in a quest to increase automation, improve network agility and security, simplify network configuration and reduce resources to establish and maintain the network. There are a number of studies exploring the development and utilisation of policies to achieve these goals. Applying the outcomes of these studies to military networks requires a clear understanding of the nature of military applications and available mechanisms to implement the appropriate policies in a software defined networking environment. In this paper, we identify several military networks where automated policy defined networking is crucial. We further present a prototype policy defined networking solution that automatically translates high-level policies into device level implementations.**

Keywords - software defined networking; policy defined networking;

I. INTRODUCTION

Military networks usually consist of a multitude of different network bearers. The core (backbone) networks are mainly connected using high-speed fibre cables or high capacity radio links (radio relays and SATCOM). Access networks (i.e., close to the end-users) are usually connected to the core using high capacity fibre links. Lower capacity radio links are used between access networks and remote end-hosts, such as forward operating bases and elements. The traditional approach to manage these networks is to individually configure a collection of devices from personal radios, switches, routers to middle boxes such as firewalls and load balancers to meet a set of high-level goals. This task is complex, tedious, and error-prone, especially in deployed environments.

Policy-based network management (PBNM) [1] 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 the intent of a user in terms of how the user wants the network management software to behave. Through a number of policy refinement steps, these policies are translated (in a completely or partially automated manner) to device configurations. PBNM presents several benefits such as requiring less manual configurations resulting in fewer errors, automated analysis and verification based on a formal foundation, dynamic checking and adaptation at runtime. PBNM was explored in [2] for military applications, especially in the context of wireless ad-hoc networks.

However, using PBNM in operational networks is not straightforward. A fully automated refinement of high-level policies down to low-level policies is still a topic of much ongoing research [3, 4]. In the last few years, significant development in software defined networking (SDN) has accelerated the development of fully automated PBNM solutions for SDN networks [5].

There is now an opportunity to apply these breakthroughs in SDN to military networks. The opportunities and challenges in applying SDN to Defence networks are eloquently provided in [6]. Although policy-based management is important for different types of military networks [2, 6, 7]. In this present work, we will concentrate mainly on networks for missions where multiple nations form a federated network as described in the NATO Federated Mission Network standard [7]. Similar standard is being developed in the US Mission Partner Environment (MPE) framework. Our contributions in this paper are:

1. An overview of military networks for multilateral missions where automated policy defined networking is important. We focus especially on the recent NATO standard for protected core and federated mission networks.
2. A prototype of a policy defined networking solution using metagraph models.

The paper is organised as follows. We discuss the background in military operations, software defined networks and policy defined networks in Section II. The metagraph model is discussed in Section III. Our prototype policy defined networking (PDN) implementation is described in Section IV. We conclude the paper in Section V.

II. BACKGROUND

A. Military Network Policies

This section briefly outlines the policies required to support military network operations. Although policy-based management is important across military networks [2, 6, 7],

we specifically discuss two recent network architectures that are being developed by the North Atlantic Treaty Organization (NATO) bodies for military operations: Mission Network (MN) and Protected Core Networking (PCN).

1) Mission Networks (MNs)

MNs are deployed on a per mission basis to interconnect different coalition partners working together in the same Area of Operation (AO). The coalition partners are also connected back to their "home networks". The focus of these networks is to create interoperability between coalition partners. An example of this is the mission network created between coalition partners in the International Security Force in Afghanistan (ISAF) [8]. The core of this network operates at an ISAF secret security level. ISAF secret security level is a mission specific security level, often on the same level as NATO secret. Directly connected to this are national networks, also at an ISAF secret security level. Through the use of information exchange gateways, these national networks are connected to their own national home networks, often at a lower NATO secret level. The gateways are middle boxes whose main function is to control information flows between the national networks (lower security level) and the mission core network (higher security level). These access controls are specified using a variety of parameters as agreed between the partners. The access parameters frequently change as the situation evolves and therefore, dynamic access control is desirable as mission requirements can change rapidly. Having an automatic policy management system would be very advantageous under these scenarios.

2) Protected Core Networking

PCN is a concept developed within the Science and Technology Organization (STO) in NATO to "implement a flexible transport infrastructure that supports military operations based on a Network Enabled Capability (NEC)" [9]. PCN allows coalition partners to interconnect and share infrastructure without losing control over their own network. Individual nations are still able to prioritize their own traffic, but at the same time are able to share idle capacity with collaborating nations. To implement PCN, network operators need to employ policies that enable them to share and reclaim network resources easily as changes occur. This is an important concept in federated military networks [7] as it allows commanders the prerogative to be able to exercise command over network under their command. It is apparent that dynamic resource allocation is important in this situation but it is often overlooked in traditional military networks. For example, consider the situation where nation A has available excess bandwidth that can be allocated to Nation B if needed. If the network path/route for nation B's traffic is required by nation A later, nation A will be able remove the resources that it established for Nation B. An important requirement in this scenario is that the establishment and removal of paths should not incur significant packet loss or downtime. In order to meet this requirement, automatic provision of dynamic Quality of Service (QoS) from high-level specifications is needed.

B. Software Defined Networking

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 that decides where packets are sent) from the data plane (the physical network that 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 obfuscating 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 abstraction is of the 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, the NBI is implemented using a programming language, e.g., NetKAT [11] or a REST/JSON API. Figure 1 presents a typical software defined network architecture.

C. Policy-Based Network Management

Policy-Based Network Management (PBNM) was developed to allow a network administrator to specify business-level goals via high-level policies [1]. The main aim of PBNM is to enable a user who is non-conversant with the technology details required to deploy a policy to manage the network. PBNM has seen commercial application in many domains: QoS management, Internet service access-control, Web-user privacy management, management of cloud architectures and others (see the survey [4] for details). In the military domain, PBNM was explored in [2].

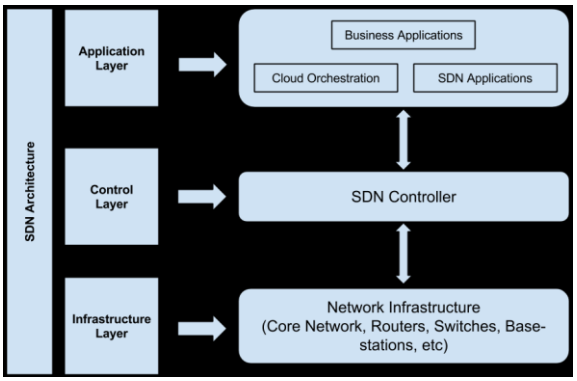


Figure 1 Software Defined Network Architecture

Non-technical managers commonly describe high-level goals using natural language. These goals are often

ambiguous and require human-translation to a machine interpretable technical specification. Thus, policy languages were required for various disciplines; some to suit specific domains like network or security management and others for general purpose: e.g., Ponder2 [12]. A review of PBNM languages for traditional, non-SDN networks is provided in [4]. Major recent developments in network programming languages for SDN have led to a number of new languages such as NetKAT [11], Pyretic [13], etc.

As policy languages mature, the research focus has shifted to software and applications that use these languages to enable fully automatic policy refinement. One important part of these efforts is policy conflict resolution. Event Calculus was used for policy conflict resolution in [14]. Event Calculus is a formal language for reasoning about policies and applies adductive reasoning to analyse policy specifications and identify conflicts. More recently, semi-ring path algebra was used in [15] for specifying and resolving conflicts in network security policies. A graphical model for policy composition and refinement in SDN networks is provided in [3].

Previous work on PBNM concentrates mostly on policies that are applied to well-defined endpoint groups of users or resources. They predominantly support policies that apply directly between a groups of users and resources without middle boxes. Furthermore, they lack the capability to analyse policies beyond simple end-to-end reachability determination. As explained above, military networks require policies that can support arbitrary grouping of users and resources and enforce predefined paths on specific traffic flows. These problems are not specific to military networks but are also common in enterprises and other types of networks [3].

Our work in this paper extends the graphical model in [3] to a more general structure of metagraphs [16]. By using metagraphs, we can model policies for data and rules management, workflow and process analysis. We show in this paper and through our prototype that metagraphs provide a rigorous and verifiable policy refinement framework for military mission networks.

III. POLICY MODEL USING METAGRAPHS

At the heart of our solution is a graph model for network policies. Our starting point is metagraph [16], a recently developed graphical structure that has been used in modelling information processing systems.

A metagraph is a directed hypergraph, which is a collection of directed set-to-set mappings. Each set is a node in the graph, with each directed edge representing the relationship between sets. Although this is a simple definition, metagraphs offer several powerful tools that can be used to analyse, optimise and troubleshoot policies [16].

In network policies, a node in the metagraph contains subset of the endpoint groups (e.g., users, time, locations), and an edge represents the mapping from one endpoint group to another. For example, an edge can represent the combination of soldiers and time into the set of "soldiers in action" at a given time. The output nodes of the metagraph represent the set of resources that can be accessed (for example, Facebook and file server).

Formally, we can view a policy equivalently as a meta-graph

MG = (Generating Set, Vertices, Edges),

where

- Generating Set is the set of all users and resources;
- Vertices represent the network endpoint groups;
- Edges represent policy constraint(s) applied to the connection between two endpoint groups.

A metagraph consists of a set of policies, where each policy is built from a set of triples (Invertex, Outvertex, Edge Attributes). Each of these triple represents a constraint in the policy. Invertex and outvertex represent the two endpoint groups and the attributes on the edge connecting the two vertices are the actions/filters we want to apply to the communication between those two ends. In other words, each constraint is defined as a triple (endpoint group 1, endpoint group 2, actions). To build a metagraph for policies, the policy designers need to provide two inputs:

- Rules to group users and resources into endpoint groups.
- A constraint on the communication between endpoint groups. For example, to allow a certain endgroup 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 identify policy conflict, we can check if there are paths from a node representing a user to two different nodes representing allowed and disallowed access to the same resource.

We have developed a prototype implementation of a metagraph based policy defined networking (PDN) solution. We present details of this prototype in the next section.

IV. PROTOTYPE OF PDN FOR MILITARY NETWORKS

Our implementation uses commodity-off-the-shelf hardware and open source implementation of SDN. The details of our policy defined networking prototype are provided in [17]. The code is publicly available at https://github.com/hxnguyen/metaSDN. We describe here the application of our prototype to military type networks.

A. SDN using OpenFlow Networks

We use embedded boards Raspberry Pi 2 model B+ running OpenvSwitch (OVS) as a commodity hardware to

implement switching functionalities of the OpenFlow protocols with the following features:

- The switch is powered by an ARM v7 quad core running at 900MHz, 1 GB SDRAM.
- It has four ports running off a USB hub. These ports are connected to USB WiFi dongles for WiFi connectivity and 4G connectivity through an iPhone.
- It runs on Debian Jessie 8 Operating System.
- OVS version 2.3.0 is installed to provide the OpenFlow switching capabilities.

An OpenFlow [10] switch is a network element that forwards packets between its ports, as defined by a remote process in the controller. Flows are broadly defined, and are limited only by the capabilities of the particular implementation. For example, a flow could be a TCP connection, or all packets from a particular MAC address or IP address, or all packets with the same VLAN tag, or all packets from the same switch port. Each flow-entry in a switch has three basic actions associated with it:

- Forward this flow's packets to a given port.
- Encapsulate and forward this flow's packets to a controller.
- Drop this flow's packets.

This simple abstraction of OpenFlow provides the basis for multiple recent advances in networking, including policy defined networking.

B. Policy Specification

We have developed a graphical user interface (GUI) for building the policy metagraph. The WEB GUI allows users to create and modify groups of network endpoints using drop-down menus. Network endpoints are identified by their IP or MAC addresses. These groups of endpoints serve as the main components in our policy compositions.

Currently, we implement only "Event-Condition-Action" policies that 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. For security applications, C is simply *allowed, disallowed, or waiting for further inspection*. For example, to specify the access control policy for mission networks that "Only coalition partners have access to the file server", the policy designer needs to select a group of users (e.g., coalition partner's staff), destination (file server) and action (allowed) from the pull-down menus.

In general, a policy is implemented using a list of "Event-Condition-Action" statements. Each of these statements is represented in metagraph as a triple of two endpoint groups and actions/filters applied to a communication between two endpoint groups. For each constraint to be added to a policy, the policy designer needs to provide two endpoint groups for the constraint and a set of operations and attributes that he wants to impose on the communication between those two end points. Currently, our prototype only supports access control attributes such as "Allow" or "Disable" communication between two end point groups, and QoS attributes such as the priority level (integer numbers) for flows between two endpoints.

Endpoint Group 1	Endpoint Group 2	A/C Action	Traffic Priority
Secure Zone	Internet	Disabled	N/A
Secure Zone	Nation 1 Zone	Allowed	Level 2
Secure Zone	Nation 2 Zone	Allowed	Level 2
Nation 1 Zone	Internet	Allowed	Level 1
Nation 2 Zone	Internet	Allowed	Level 2

Figure 2 an example of access control policy with different security zones

Figure 2 illustrates an example of access control policy in a mission network. As the example shows, the policy is essentially a list of five constraints (represented by the five rows). Secure Zone, Nation 1 Zone, Nation 2 Zone and Internet are groups of endpoints that are previously created by the policy designers and selected during the policy composition. Once this policy is deployed to our switches, endpoints in Nation 1 Zone and Nation 2 Zone will have access to Internet endpoints, however the allocated traffic bandwidth for each is different in which Nation 1 Zone has higher priority than Nation 2 Zone. Furthermore, the Secure Zone is only accessible from Nation Zones but does not have any connection to endpoints in Internet.

From these specifications, the analysis engine will build a metagraph for verification and checking. After checking for correctness using metagraph algorithms, the engine will auto-generate OpenFlow statements from the high-level policy specification above. We explain the next two steps of verification and OpenFlow generation in the next sections.

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 common specify conflicting actions or attributes, then there will be a conflict. In order to detect such conflicts from our policy metagraph, we perform graph analytic algorithms on the resulting graph to detect possible conflicts.

For example, consider a policy in Figure 3 with three triples of (Invertex, Outvertex, and Attributes). Each of these triples represents a constraint on the communication between two endpoint groups of end hosts (h1, h1, etc) or resources such as web servers (ws1 and ws2).

D2	W1	Disabled	N/A
D1	W2	Allowed	Level 2
D3	W1	Allowed	Level 2

Figure 3 Example policies with conflicts. There are five endpoint groups D1={h3, h5, h4}, D2={h1, h2, h3}, D3 ={h2, h3, h5}, W1={ws1, ws2} and W2 = {ws2, ws3}.

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 are very common in access control policies specified on groups of endpoints [5,12].

The metagraph for the policy in Figure 3 is shown in Figure 4. Using this metagraph, detecting conflicts becomes an exercise of reachability analysis on metagraph [16]. We present an implementation of such algorithms. Many other advanced analyses on metagraphs are provided in [16]. In order to detect conflicts in a policy, we use the following steps:

- Normalise the metagraph and turn it into a flat or plain graph where common parts of the original metagraph vertices will be extracted and form disjoint subsets that then serve as vertices in a normal graph. The normalisation process partitions the original meta vertices into non-overlapping subsets. Take the metagraph in Figure 4. The end point groups D1, D2, D3, W1, W2 are broken down into subgroups of atomic elements {h0, h1}, {h2}, {h3}, {h4}, {h5}, {h6}, {ws1}, {ws2}, {ws3}. Atomic elements allow automatic conflict detection and OpenFlow code generation as explained below.
- Edges of the metagraph 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").

The above algorithms have linear runtime. The Python code to separate vertices in a metagraph into globally disjoint subsets is shown in Figure 5. The algorithm works by iteratively looking at their intersections among vertices. Once the disjoint groups are identified, re-assigning edges to these subsets can also be done in linear time by iterating through the list of original edges. Note that in this new graph, there could be multiple edges between two nodes.

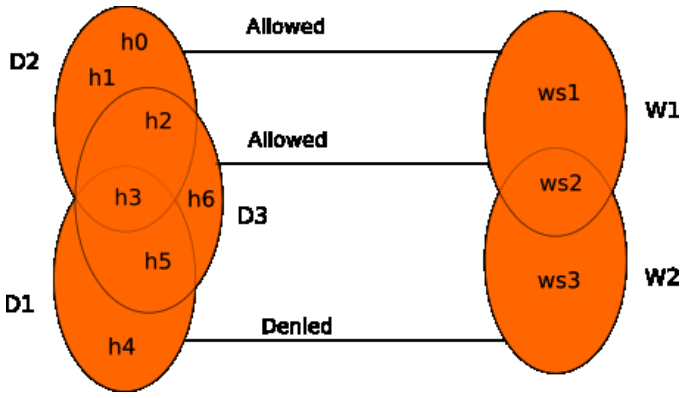


Figure 4 Metagraph for the example policy with 5 endpoint groups and 3 constraints.

Our current implementation only supports actions such as "Allow access"/"Disable access" and the level of traffic

```
my_dict = dict()
SEPARATOR = "_"
for vertex in Metagraph.vertices:
  for element in vertex.elements:
    if self.my_dict.has_key(element):
      self.my_dict[element]    +=    SEPARATOR    +
str(vertex.id)
    else:
      self.my_dict[element] = str(vertex.id)
        disjoint_sub_sets_identifiers = my_dict.values()
```

Figure 5 Python code for identifying disjoint endpoint groups

priority. 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, this function will need to be extended. 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 policy designer 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.

See [15] for more discussions on the algebras and shortest path algorithms.

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 currently optimising the number of OpenFlow statements. As a result, the deployed OpenFlow code may not contain the smallest possible number of OpenFlow statements. We plan to investigate this optimisation step in our future work.

E. QoS Management: Policy Traffic Prioritisation

We have also implemented policies for traffic prioritisation of flows from different sources as needed in PCN context. To implement traffic prioritisation for a flow, we label the edge connecting the source and destination groups with the label "high" or "low".

Once the prioritisation 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, if the flow between {h1} and {ws1} is assigned a ``high" traffic priority, we map the flow to queue 1 (90% of bandwidth).

Note that for every pair of source and destination endpoint groups, all the links in the path between the groups have to be set with the same priority level. More specifically, for each constraint in the policy, we have two sets of endpoints (say X and Y) and the traffic priority TP applied to this communication. That is for every pair of endpoints (x,y), we have a pair of IP addresses and a traffic priority level of TP. The analysis engine maps the constraint into OpenFlow statements for all switches on the path from x to y. These statements will then be forwarded to the POX Controller as a JSON object that contains IP addresses of x and y and a queue number corresponding with value of TP. Here we are assuming that the OpenvSwitch already has queues set up at each of its ports and that this setting is known to the controller beforehand to facilitate mapping between queues and traffic priority levels.

V. CONCLUSION

In this paper we presented a policy defined networking framework using metagraph models, for military networks. Our focus is on mission networks and protected core networks as defined in recent NATO standards. We show that these networks can greatly benefit from the rigorous mathematical policy framework provided by metagraphs. Therefore, our prototype PDN solution holds great promise for enabling error-free, agile and secure network management and control in future military networks.

ACKNOWLEDGEMENT

This research is supported by the Australian Research Council Linkage Grant LP100200493.

REFERENCES

[1] G. Waters, J. Wheeler, B. Moore, A. Westerinen, and L. Rafalow, "Draft IETF Policy – Policy Framework Architecture", Feb. 1999. [Online]. Available: https://www.ietf.org/proceedings/46/I-D/draft-ietf-policy-framework-00.txt.

[2] R. Chadha, "Beyond the Hype: Policies for Military Network Operations," Systems and Networks Communications, 2006. ICSNC '06. International Conference on, Tahiti, 2006, pp. 38-38.

[3] Chaithan Prakash, *et al.*. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (SIGCOMM '15). ACM, New York, NY, USA, 29-42. DOI=http://dx.doi.org/10.1145/2785956.2787506

[4] Weili Han and Chang Lei. 2012. Survey Paper: A survey on policy languages in network and security management. Comput. Netw. 56, 1 (January 2012), 477-489. DOI=http://dx.doi.org/10.1016/j.comnet.2011.09.014

[5] OpenStack Project, "Group Based Policy Abstractions for Neutron". [Online]. Available: https://wiki.openstack.org/wiki/GroupBasedPolicy. Viewed 06 June 2016

[6] Stephan Lawson, "Networks Need Automation – just ask the U.S. military", NetworkWorld, 11 May 2016. [Online]. Available: http://www.networkworld.com/article/3068662/networks-need-automation-just-ask-the-us-military.html

[7] "Federated Mission Networking (FMN)." http://www.act.nato.int/fmn, viewed 18 August 2016.

[8] Erik Sørensen , "SDN used for policy enforcement in a federated military network", Master Thesis, Norwegian University of Science and Technology, June 2014.

[9] G. Hallingstad and S. Oudkerk, "Protected core networking: an architectural approach to secure and flexible communications," Communications Magazine, IEEE, vol. 46, no. 11, pp. 35–41, 2008.

[10] McKeown, N, Anderson, T, Balakrishnan, Parulkar, H, Peterson, G L, Rexford, J, Shenker S & Turner J 2008, "OpenFlow: enabling innovation in campus networks", SIGCOMM Comput. Commun. Rev., vol. 38, pp. 69–74, Mar. 2008.

[11] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: semantic foundations for networks. SIGPLAN Not.49, 1 (January 2014), 113-126. DOI=http://dx.doi.org/10.1145/2578855.2535862.

[12] Kevin Twidle, Naranker Dulay, Emil Lupu and Morris Sloman, "Ponder2: A Policy System for Autonomous Pervasive Environments", In Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems, April 2009.

[13] Christopher Monsanto and Joshua Reich and Nate Foster and Jennifer Rexford and David Walker, "Composing Software Defined Networks", Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013.

[14] A. K. Bandara, E. C. Lupu, A. Russo, "Using Event Calculus to Formalise Policy Specification and Analysis", 4th IEEE International Workshop on Policies for Distributed Systems and Networks, Como, Italy, 2003.

[15] D. Ranathunga et al., "The Mathematical Foundations for Mapping Policies to Network Devices (Technical Report)", 2016.

[16] Amit Basu and Robert W. Blanning. 2006. Metagraphs and their Applications (Integrated Series in Information Systems). Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[17] Nguyen et al., "Toward Policy Defined Wireless Access Networks", Technical Report, The University of Adelaide, 2016. [Online]. https://github.com/hxnguyen/metaSDN/blob/master/metaSDN.pdf