Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security

Sungmin Hong
SUCCESS Lab
Texas A&M University
ghitsh@tamu.edu

Robert Baykov
SUCCESS Lab
Texas A&M University
baykovr@cse.tamu.edu

Lei Xu
SUCCESS Lab
Texas A&M University
xray2012@cse.tamu.edu

Srinath Nadimpalli
SUCCESS Lab
Texas A&M University
srinathnadimpalli@tamu.edu

Guofei Gu
SUCCESS Lab
Texas A&M University
guofei@cse.tamu.edu

***Abstract*—An emerging trend in corporate network administration is BYOD (Bring Your Own Device). Although with many advantages, the paradigm shift presents new challenges in security to enterprise networks. While existing solutions such as Mobile Device Management (*MDM*) focus mainly on controlling and protecting device data, they fall short in providing a holistic network protection system. New innovation is needed in providing administrators with sophisticated network policies and control capabilities over the devices and mobile applications (apps). In this paper, we present PBS (Programmable BYOD Security), a new security solution to enable fine-grained, application-level network security programmability for the purpose of network management and policy enforcement on mobile apps and devices. Our work is motivated by another emerging and powerful concept, SDN (Software-Defined Networking). With a novel abstraction of mobile device elements (e.g., apps and network interfaces on the device) into conventional SDN network elements, PBS intends to provide network-wide, context-aware, app-specific policy enforcement at run-time without introducing much overhead on a resource-constrained mobile device, and without the actual deployment of SDN switches in enterprise networks. We implement a prototype system of PBS, with a controller component that runs a BYOD policy program on existing SDN controllers and a client component, PBS-DROID, for Android devices. Our evaluation shows that PBS is an effective and practical solution for BYOD security.**

I. INTRODUCTION

BYOD is the new paradigm in the workplace. The enterprise is facing growing limitations of traditional infrastructure, rising cost overheads, and a slowing pace in adopting new technology. In lieu of utilizing company resources to keep up with the torrential downpour of new devices and gadgets year after year, BYOD enables the enterprise to leverage employee-owned devices in the workplace. This fresh concept, *bring your own device*, shifts the cost overhead of device ownership away from the corporation. No longer do administrators and infrastructure managers handle acquisition of new hardware. The tempting offer of offloading the cost of device acquisition onto the employee has seen a rapid growth in adaptation amongst enterprise networks. In fact, studies in 2012 [16] have shown that 44% of users in developed countries and 75% in developing countries are now utilizing BYOD in the workplace. This adoption rate shows no signs of slowing, and we cannot ignore this paradigm shift.

The crux of BYOD, however, is its close and convoluted interplay with network security. While some of the *physical* devices have changed ownership, the role of management remains firmly in the realm of corporate administrators. *Mismanagement* by the users potentially opens the workplace to an array of unwanted or even malicious applications. Providing device security, traditionally fostered by administrators, is now made significantly more complex in BYOD. Today, the challenge for administrators is the management of dynamic BYOD-enabled devices and the diverse apps running on them. Migrating to-and-from work, this new breed of corporate device leaves the safe haven of the company at the end of the workday. These devices require additional security against outside threats. More importantly, the corporate network itself now needs more security and management capabilities to handle its devices. Without proper management tools, the benefit provided by BYOD is overshadowed by the holes created in the enterprise security environment.

Existing solutions have sought to plug the gap in device management through on-device administration and network-wide device management systems. For example, Google has provided such a system, Android Device Administration (ADA), as early as Android 2.2. This system seeks to provide management features via APIs [2], which facilitate the creation of security-aware applications on the device. Capabilities such as strong password policy enforcement and remote device wiping are natively provided by *ADA*. However, today's complex deployments require more features and dynamic reconfigurability in order to effectively manage and secure the evolving network. Recent systems, such as Mobile Device Management (*MDM*) solutions proposed in [3], [5], [6], [8], [4], have come to supplement *ADA* and have led the technical trend in BYOD management. These solutions provide additional granularity and complexity in management capabilities. More recently, Android for Work [1] introduced a dedicated "Work Profile" to separate corporate and personal application data for BYOD deployments. The profile supports OS-level encryption and additional management APIs to third-party MDM/Enterprise Mobility Management (EMM) partners via *ADA*. This enabled administrators to build new management solutions for employee devices.

However, these solutions focus mainly on device/app data control and protection (e.g., through domain isolation as in Samsung KNOX [7] and Android for Work [1]). They lack sufficient network management features, such as fine-grained network security policy enforcement or access control, which are imperative to a comprehensive enterprise device management scenario. One practice may attempt to apply existing security policies to new BYOD devices using extra network access control solutions/infrastructure. Yet simple adoption of traditional solutions is not well-suited to reflect dynamics of BYOD devices.

Complementing existing work, our approach focuses on addressing the issues posed by BYOD in the context of mobile devices, such as phones and tablets. Our intent is to provide a new system for BYOD management which features dynamic programmability and security policy enforcement with unprecedented granularity. Different from existing solutions, we do not aim to control user actions or the device storage implicitly. Rather, we wish to secure and manage the *network access capabilities* of the BYOD device, with high fidelity such as *individual-application-specific* and *device-context-aware* network access. We seek to provide such detailed access control in a sophisticated and easy-to-use system, providing a holistic network-wide management platform.

A unique requirement in today's mobile BYOD networks is the management of the dynamic network devices. In this case, the device may physically move or change its run-time contexts (e.g., add/remove applications) during its operation. While the device exists on the network, administrators require real-time adaptability and control of device network actions. For instance, while an administrator wishes to restrict network communication between specific apps/devices/network-resources, they may not possess intricate knowledge of all applications which enable such functionality. It may be useful to first learn device communication behavior and context information (e.g., location, time) and then programmatically apply a policy which limits unwanted capability in the workplace. Thus, an administrator requires additional capabilities to complement existing management systems. Such capabilities require rich granularity and dynamic configurability, which are difficult to provide in existing, mostly static management systems.

To address these dynamic network management concerns, we propose our system, PBS (Programmable BYOD Security), a new security solution to bring fine-grained, programmable network policy enforcement to BYOD devices in enterprise networks. Our solution is inspired by the concept of Software-Defined Networking (SDN), which provides a new networking architecture to enable network-wide visibility, programmability, and control. Different from existing SDN techniques that require revolutionary changes and replacement of the network equipment infrastructure, such as routers and switches, PBS applies the SDN concept to the mobile-device level hardware, and does not require any changes to the existing network infrastructure (i.e., PBS does not require the actual deployment of SDN/OpenFlow network switches). Analogous to conventional SDN, yet tailored to mobile environment, we abstract a user's mobile device as a logical switch, with apps running on the device as logical hosts and all the available network interfaces as logical ports. We map the user's context and mobile device management features to functionalities of SDN. We extend conventional SDN flow control capabilities to perform fine-grained app-level, context-aware, dynamic, programmable policy control. Our remote PBS controller can run a user-defined security policy program that controls and monitors application-aware flows with the user's contexts at the mobile device level at run-time; thereby, all the flows between applications and device network interfaces are visible and programmable with ease. Additionally, in order to facilitate ease of use of the remote controller, PBS introduces a rich policy language for the configuration and management of BYOD devices. The policy language enables the enterprise administrator to specify device management decisions without modifying the PBS controller program, thus allowing the easy administration of PBS and its management devices without intricate knowledge of SDN, lowering the barrier of entry for real world use cases.

In summary, this paper makes the following contributions:

- We propose a novel two-tiered network-wide policy enforcement scheme to control BYOD devices based upon SDN/OpenFlow techniques. Different from existing architecture of SDN/OpenFlow, our approach leverages novel transparent application abstraction, efficient flow-policy architecture, and optimization schemes, which are tailored to mobile BYOD devices. Our solution empowers global visibility and flexible security programmability to enterprise BYOD network administrators as well as reactive policy update in Android devices.
- We present dynamic and fine-grained access control over context-aware flows at a mobile application level. In particular, we dynamically enforce access control logic over application flows by considering the run-time contexts of mobile users.
- We design and implement a PBS client prototype system on Android, called PBS-DROID, as well as an enterprise network PBS controller instantiated in an SDN controller. We choose Android because it is open-source and dominating the mobile device market. Our techniques should be applicable to other mobile platforms, or even the regular PC platform.
- We evaluate the effectiveness and performance of PBS-DROID with real-world Android applications. The results show that PBS-DROID introduces a negligible performance overhead and minor impact to battery life, while achieving desired policy control functions for BYOD.

The rest of the paper is organized as follows. Section II introduces the background of SDN/OpenFlow and problem statement. Section III provides design details of our PBS solution. Performance evaluations with use cases are presented in Section IV and further discussions are addressed in Section V. We review related work in Section VI and conclude the paper in Section VII.

II. BACKGROUND & PROBLEM STATEMENT

In this section, we first briefly review the background of SDN. Next, we motivate the need for a dynamic, programmable system for the granular management of BYOD

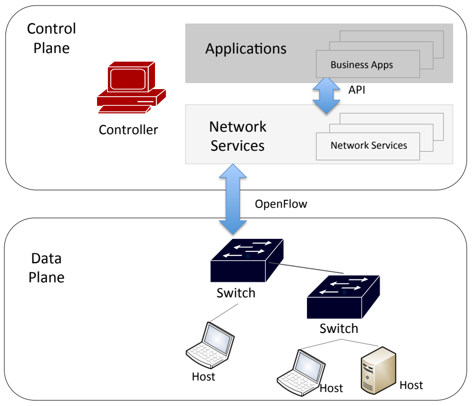


Fig. 1: The Concept of SDN/OpenFlow

apps in enterprise networks. We then present the research questions in our problem space.

A. SDN/OpenFlow Background

Software-Defined Networking (SDN) has emerged as a promising technology to innovate the ossified network infrastructure. This paradigm decouples the control and data planes, thus allows for sophisticated and flexible control plane traffic management. As the most widely used realization of SDN, OpenFlow defines an interface between the control plane and the data plane.[1]

Figure 1 depicts the operation of an SDN/OpenFlow network. A switch contains simple forwarding fabrics in the data plane (for physical ports) and forwarding rules for ingress packets to look up where to transmit the packets among the ports. If a flow rule matches a specific packet, the data plane executes one of the following actions: (i) forward the packet, (ii) drop the packet, or (iii) send the packet to the control plane. If no rule matches, the switch sends a new flow request (i.e., Packet-In message) to the controller for a flow rule installation. The control plane manages the data plane by instructing the installation of new flow rules via Flow-Mod messages. The control plane keeps track of the physical information, such as ports of connected switches via Port-Status messages. Such physical information facilitates the flow rule decision-making process. Additionally, the control plane maintains a holistic view of the network by requesting and retrieving link layer discovery protocol messages (LLDP) which aid in the construction of network topology. Lastly, messages such as request/reply (i.e., Stat-Request/Reply) are used to check the statistics of traffic and Packet-In messages. The messages provided by the control plane are, in turn, used by control applications which provide higher order logic for network control. The applications consist of programmable, software logic which provides SDN network management services. The boundary between the network applications and the control plane protocol messages is commonly referred to as the Northbound API.

B. Motivating Examples

A commonplace BYOD scenario grants employees the ability to utilize their personal devices in the workplace. By doing so, the employer benefits in a reduced cost overhead associated with physical device acquisition. However, supporting employee-owned devices creates new issues in network administration and security.

Application Awareness & Network Visibility. A key issue in BYOD enterprise network administration is the maintenance of a holistic view of devices and their application behaviors on the network. Although it is possible to observe basic device connection activities via traditional network monitoring tools. Advanced information, such as device context and detailed application-connection information is invisible to traditional tools. For example, an application could send data through a network interface, such as 3G/4G interfaces, normally not visible to the enterprise network administrator. Furthermore, it is currently not possible to correlate application network activities with the hosting device's physical location. Such physical information is highly useful in crafting detailed management policies in the enterprise, but cannot be discerned by traditional tools.

Additionally, the issue of enforcing network-wide security policies on BYOD devices is of critical importance. For example, a scenario in which the administrator attempts to restrict access to company resources, such as a data server, only to a specific *enterprise-allowed* application while a device is present in the server room. Access restriction by devices could be implemented by traditional infrastructure via a static scheme, e.g., a network-wide firewall rule. However, it has two limitations: First, existing methods cannot account for device and application context, e.g., application context, mobile location, and network flow information specific to the device/application; Second, static policies cannot react to changing network dynamics. For instance, events such as new devices entering the network, device context updates or suspicious activity analysis all require additional logic beyond static configuration.

Dynamic Policy Programming. The ability to provide dynamic policy programming/updates is a key motivation in our work. For example, a network administrator may wish to provide application-specific time and location context restriction, such as allowing social media applications only during lunch hours *and* in areas designated as break areas during predefined hours. It would also be advantageous to specify dynamic policies, capable of redirecting malicious/suspicious traffic for further inspection or quarantine at security middle boxes, e.g., in the case of a device/application which suddenly exhibits suspicious network behavior.

C. Research Questions

The aforementioned functions require detailed, fine-grained device and application monitoring not possible in existing solutions. Motivated by the issues outlined above, we discuss the following research questions.

- **Can we use traditional solutions?** Traditional security solutions (e.g., ACLs/firewalls) are difficult and inflexible to program, deploy, and manage in dynamic, network- and application-aware security policy enforcement for BYOD scenarios. Also, traditional access control policies are typically coupled with physical devices/resources instead of applications. We

[1]In this paper, we use SDN and OpenFlow interchangeably.

need a new security framework to handle BYOD-specific access control management.

- **Can we apply the legacy SDN infrastructure?** The SDN infrastructure requires network devices (i.e., switch and router) to be SDN-enabled. While the controller can be deployed on existing commodity hardware, underlying network devices must still be SDN-enabled in order to construct programmable/SDN networks. For large networks, upgrading or switching to SDN-enabled hardware can be costly. Even if we assume that the organization is already armed with SDN infrastructure, it is still insufficient to enable fine-grained application control. This is because existing SDN networks have difficulty in distinguishing the source/destination of application packets to/from network devices. Furthermore, many modern BYOD devices are equipped with multiple physical/logical network interfaces capable of connecting to different types of networks (e.g., 3G/4G, private networks using WiFi direct or device pairing). This leads to a loss of global network visibility in the enterprise network because legacy SDN controllers do not manage traffic on such additional networks/interfaces.
- **How much granularity we should provide?** Existing network information such as layer 3 and above is required to implement basic security policies. However, this information is not sufficiently granular for providing advanced security policies. For example, it is necessary to include layer 2 header fields in order to inspect and control L2VPN and VLAN tags. Such information is useful to network administrators and security middle-boxes such as intrusion detection systems (IDS). Moreover, network flow information alone does not provide a full picture of BYOD devices. In order to gain a holistic view of device activity we need to couple application information and application-aware user contexts with network flow information.

III. System Design and Implementation

In this section, we present the design and implementation of our PBS system, which mainly includes PBS-Droid, an Android version of PBS client, and PBS Controller. With consideration for the issues described in Section II, we first summarize our design goals as follows:

Fine-grained Access Control. Our system aims to provide fine-grained access control to application-specific network flows and extraction of rich application-specific details such as context and layer-2 information.

Dynamic Policy Enforcement. Our system aims to provide enterprise administrators with the ability to enforce dynamic access control at run-time based on application-specific policy and network behavior.

Network-wide Programmability. Our system aims to provide a programmable network-wide policy enforcement system to enterprise network administrators.

Minor Performance Overhead. Targeting mobile devices, our system aims to minimize performance overhead and resource consumption.

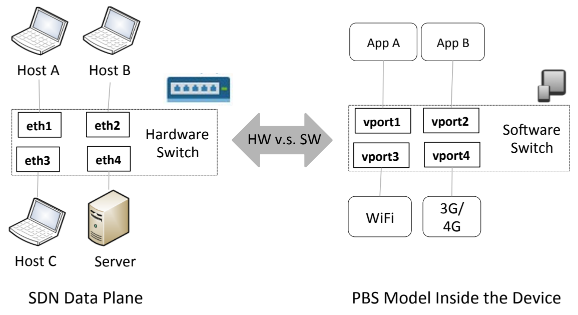


Fig. 2: Abstraction of PBS-Droid with the SDN concept

A. Trust Model

In this paper, we assume that the enterprise network is trusted and our system is installed/enforced on BYOD Android devices previously vetted and authenticated by the administrator. As is common practice in existing *MDM* solutions ([1], [3], [5], [6]), the device is only used according to the terms of use and privacy agreement of the corporate BYOD program. In addition, the network traffic from the users' devices is monitored by IT admins only at the work place as per user agreement and apps run inside the separate domain (e.g., work profile in Android for Work and container in KNOX) for enterprise use. For the purposes of our system design, we assume that the Android framework, libraries, and the kernel layer of the Android device are not compromised, and the Android OS is trusted. We also assume that the device does not have root privileges. This assumption is fully valid given that all hardware vendors disable root privileges on devices prior to distribution. We also assume that the device has several other apps that the user has installed in a private domain other than enterprise-allowed apps. In addition, the communication between PBS-Droid and PBS Controller is secured (e.g., by SSL). PBS does not deal with the security risks related to data encryption and storage, which are handled by existing complementary MDM solutions. Also, some controller-specific deployment and scalability issues (e.g., physical/logical configurations/distributions, distributed control planes, high availability) are out of the scope of this paper.

B. Modeling and Coupling Inside the Device

In this section, we describe the core design concept that facilitates our system's application-flow management on the mobile device. We draw a parallel between the SDN data plane switch and our design, which embraces the concept of a "virtual" switch as shown in Figure 2. While an SDN data plane facilitates communication amongst a set of network devices via a port-host mapping, a "virtual" switch provides communication between *virtual ports* and software-entities. To enable application-flow management, we treat all mobile device applications and network interfaces (e.g., WiFi, 3G/4G) as network port entities on a virtual switch. By mapping applications and network interfaces to unique virtual ports, we enable flow management of all application network traffic inside of our virtual switch. This allows for easy flow management and efficient application flow isolation, as well as the utilization of existing SDN concepts, which readily function in data plane switches. Furthermore, we couple device context information, such as time and GPS location, with each

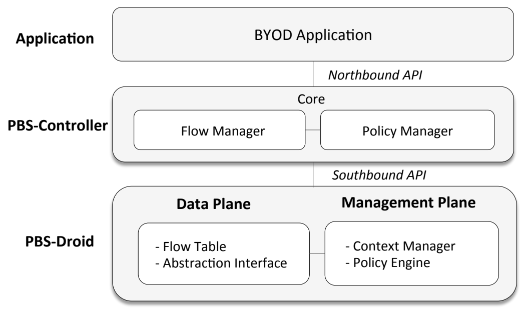


Fig. 3: High Level Architecture

application network flow. In this way, the controller is granted access to not only application network flow information but also the *context* of the application during network activity. This enables the controller to perform advanced decision making with fine-grained connection and context information on a per-app basis for each managed mobile device.

C. Architecture and Operation

As illustrated in Figure 3, our system consists of three key components: PBS CONTROLLER, PBS-DROID, and BYOD application.

High Level Composition. First, PBS CONTROLLER provides the core features needed to facilitate administration actions such as policy definition and enforcement. Two core controller sub-components, *Flow Manager* and *Policy Manager*, provide administrators with expanded management operations. *Flow Manager* provides transparent flow management, operating much like existing SDN controllers. While the *Policy Manager* implements BYOD-specific policy management which cooperates with the *Flow Manager*. The access to the managers is exposed by the Southbound and Northbound APIs to PBS-DROID and controller applications respectively. The Southbound APIs provide interfaces for registering, removing, and modifying flow-rules and policies by PBS-DROID. The Northbound APIs provide interfaces to define flow rules and policies, flow handlers, and policy protocol handlers via controller applications.

Second, we provide a BYOD controller application layer which levies the Northbound APIs to provide a programmable interface for network management operations to the administrator. The BYOD application allows operations such as defining flow rules and policies as well as monitoring and analyzing flows/packets and running security functions.

Third, PBS-DROID contains Data Plane which is composed of a flow table and network interfaces just as in the standard SDN architecture, however, PBS-DROID also includes additional mappings which identify Android applications and interface bindings as discussed in Section III-B. The Management Plane component monitors and manages device contexts, such as time and location. The Policy Engine maintains the policy table and executes conditional policy actions according to the device contexts at run-time. For example, the Management Plane will notify the Policy Engine of a device location update, which in turn may make a new policy action by installing a device/app flow rule.

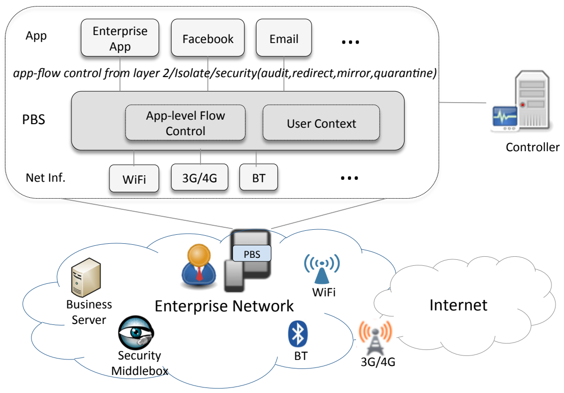


Fig. 4: Operations of PBS

In total, the systems which comprise PBS provide an extension to the Android platform to run dynamic, fine-grained, context-aware policy enforcement and network access control.

Sample Operation. To show an example of the operation of PBS, suppose that we have a device with two applications installed, a non-business app[2] and an enterprise business app as illustrated in Figure 4. The network administrator specifies a policy to allow only the enterprise business app to access a company network via standard wireless interface (WiFi) while the employee is at work with the device. The controller enforces this policy by instructing PBS-DROID to install flow rules on the user device which restrict all user app's network communications while the device is in the workplace location context. When the device leaves the workplace it resumes normal operation, as common practice in other systems such as KNOX [7] which provides a work and home mode.

This simple example demonstrates the basic usage of our system, however, much more sophisticated scenarios with dynamic device context updates and programmable policies are possible. For example, because PBS can monitor and manage extremely fine-grained layer-2 information, we may implement a VLAN tag on an application specific basis, isolating application network traffic without modifying device network configurations. The application needs not be aware of our modification, in its scope of operation the packets continue to arrive normally. Furthermore, a set of devices can be dynamically placed on an L2VPN (virtual private network) during a context switch, such as a location change from one building to another in the workplace. Lastly, the controller can monitor all application flows active on the network at real-time, thanks to the fine granularity of each flow. The controller can reactively issue rules redirecting app traffic based on activity, such as suspicious actions like network scanning. Such features/actions are inspired by existing SDN security work [24], [22], [20], [21] which addresses network threats using SDN.

D. System Internals

PBS-DROID incorporates three components from a functional perspective: Abstraction Layer, Data Plane, and Management Plane (illustrated in Figure 5).

[2]We assume that a user is allowed to install and run non-business apps under enterprise permission.

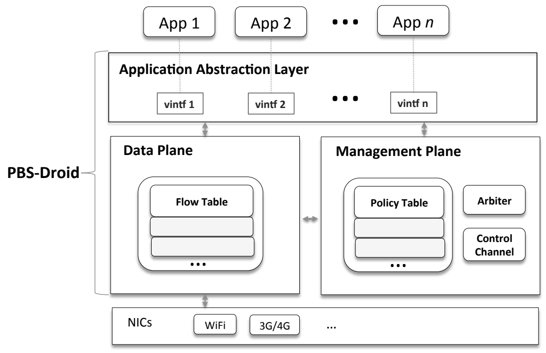


Fig. 5: System Components

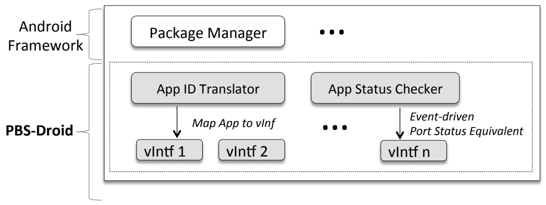


Fig. 6: Application Abstraction Layer

1) Enabling Fine-grained Access Control: In order to provide application-aware flow control with layer 2 granularity, we present an Abstraction Layer and Data Plane as shown in Figure 5.

Abstraction Layer serves two roles: (1) collecting application-level semantics for PBS CONTROLLER, (2) adopting application-level control logic from PBS CONTROLLER. For application-level semantics, we obtain a user identifier (UID), a package name for each application on a device, which is extracted from the Android *PackageManager* at the framework layer as shown in Figure 6. The UID serves the purpose of identifying application flows, this is because the flow-connection-information is not correlated with other application unique identification information. The package name identifies an application by name, however it is a long character string and sending such a long name in a packet may unnecessarily reduce system performance. Thus, we employ a hashing mechanism to convert the string to an unsigned integer. The UID and package name hash are then sent to the Data Plane for further processing.

Upon receipt of the arbiter message containing the UID and package name hash, the Data Plane creates an internal virtual interface using another hash value, calculated from the concatenation of the UID and package name hash. The virtual interface is then added to the bridge of the datapath, via the standard Linux function. Thereby, the bridge-registered virtual network interface name uniquely ties both the Android package name *and* the Linux network stack connection flow. When the status of an application changes, App Status Checker sends the change of states to the Data Plane and the status is interpreted to the status of corresponding virtual interface accordingly.

This abstraction scheme enables us to treat an application-specific virtual interface as a normal network port with great ease. That is, application attributes are informed to the controller during a secure connection handshake through a *Port-Status* message and the equivalent status of a virtual interface is reported to the controller seamlessly. As a safeguard, a fingerprint of the key used to sign the package may be sent to the PBS CONTROLLER for the purpose of application validation, which allows for discerning of possible duplicate identifiers which may arise from the application-level information extraction.

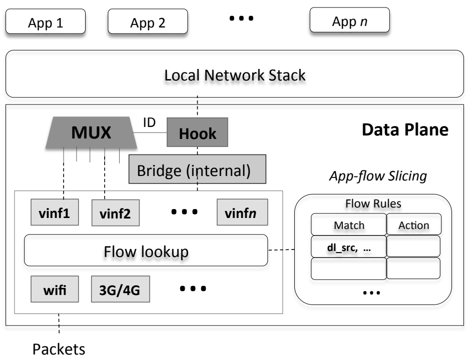


Fig. 7: Data Plane

Data Plane is the forwarding fabric that manages fine-grained flows inside PBS-DROID. The functionality of the Data Plane stems from its maintenance of flow tables which contain flow rules, which are used for the enforcement of access control logic as shown in Figure 7. The internal bridge (layer-2 interface) is linked to the local network stack and contains both physical and virtual interfaces in order to capture layer 2 frames. When an interface receives packets, the Data Plane checks the flow table for matching flow entries. If there is a match (from layer 2), it executes the corresponding actions (e.g., forwarding, drop, controller); otherwise, a new flow request message is sent to the controller for further flow decision making. We have two types of flow tables to store flow entries, one in user space and the second in kernel space. The kernel space table supports fast match lookups via a kernel cache, while userspace table allows full match lookups using a proactive tuple space search. Besides the normal packet processing from a physical network interface, an additional mechanism for handling application-aware flows through application-specific virtual interface is required. This is because in order for packets from an application to pass through the corresponding virtual interface, we must hook the packets before they reach layer 2 and forward them to the corresponding interface. Here, we utilize *netfilter* to hook packets after the local network stack completes packet processing. To do so, we install the *netfilter* hook inside the data plane and capture the packets at *NF_POST_ROUTING* level. When a packet is hooked, we extract a UID from the socket buffer (*struct sk_buff*) and look up a matching package name and UID. Then, we forward it to its mapped virtual interface through the mapper which is generated in the Data Plane via the abstraction layer. The operation is done by redirecting the *netdev* pointer of *sk_buff* in the kernel module. This mechanism enables us to forward an application packet to the virtual interface in a much faster and more effective way than using the standard user-space network commands such as *ip, route*. On the other hand, the reverse direction of a packet from the network interface to an app is managed by the controller by using a flow rule. The application-aware flow

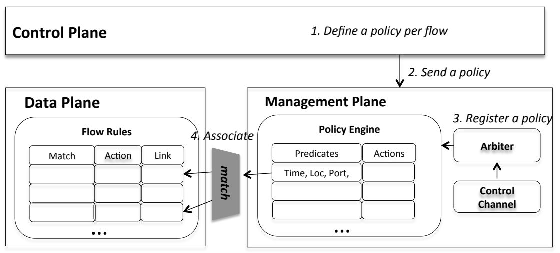


Fig. 8: Policy Registration in Management Plane

TABLE I: Policy Protocol

Message	Type	Direction	Description
OFPT_FEATURE_REPLY	OF	From Device	Device Info
OFPT_PORT_STATUS	OF	From Device	App Status
PBS_REGISTER_POLICY	PBS	To Device	Add Policy
PBS_REMOVE_POLICY	PBS	To Device	Remove Policy
PBS_MODIFY_POLICY	PBS	To Device	Modify Policy
PBS_REPORT	PBS	From Device	Result, Info

management scheme provides an application-flow slicing by the controller so that app-flows can be easily isolated among applications by the flow rule.

2) *Enforcing Dynamic Context-Aware Policy:* In this section, we describe the *Management Plane* which facilitates enterprise policy storage, device context update lookup, and PBS CONTROLLER communication. It consists of three components residing on the mobile device: Policy Engine, Arbiter, and Control Channel.

The Policy Engine maintains an administrator-defined policy table on the mobile device. Note that entries in the table are not specified by user, instead they are composed by the PBS CONTROLLER from our High-Level Policy Language (shown in Figure 10) discussed in Section III-E. Each composed entry in the policy table consists of three parts: predicates, actions, and a match field. Predicates are a set of multiple conditions separated by conjunction, e.g., time is noon *and* location is building a. Actions are associated with flow rules for policy enforcement. For example, an action may modify the corresponding flow rule or packet header fields in Data Plane to redirect, mirror, or quarantine the flow. This scheme allows us to tightly couple a flow with a user-context based policy, which is not supported by existing SDN/OpenFlow. Lastly, a match field is used to identify policy to flow rule associated. The match field allows a wildcard in order for multiple flows to be associated with a single common/global policy entry, which saves the size of the policy table.

The Arbiter provides the function to retrieve Android device context, which is done in real time by listening to predefined PBS-DROID relevant events, detailed in Table II. The Arbiter serves the function of monitoring device specific context updates and sending them to the Policy Engine. When a policy entry predicate is satisfied with a context event, the corresponding policy action is carried out. Figure 8 depicts the procedure of the policy registration from Control Plane to Management Plane via the Arbiter. The Arbiter not only causes the Policy Engine to perform flow-control actions in response to context changes, but triggers the engine to notify the PBS CONTROLLER of device contexts according to the policy via the control channel.

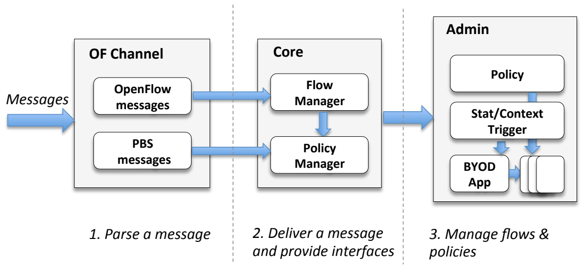


Fig. 9: PBS CONTROLLER Flow Diagram and Interfaces

The control channel facilitates the secure communication between the Management Plane and PBS CONTROLLER. The connection is secured via SSL/TLS and serves two purposes. First, the control channel allows for basic OpenFlow protocol messages, such as flow rule updates and controller decision queries. Second, the control channel processes our policy protocol (as shown in Table I) allowing bi-direction communication between the controller and PBS-DROID for policy management. The protocol borrows from OpenFlow in its design, directly utilizing two existing OF messages without modification and adding four new PBS specific messages. This is because we encode application information into the virtual interface port name, feature reply and port status messages allow us to query for the port name and status respectfully, yielding application information transparently.

E. PBS CONTROLLER

In this section, we describe the PBS CONTROLLER which provides a central programmable interface to the network administrator. Our controller design involves utilizing an existing SDN controller with new extensions, detailed as follows.

Network Programmability. As previously stated in Section III-C, the controller provides functions via Northbound APIs for network-flow and policy management. These API function calls are coded into the BYOD application in the controller implementation language (e.g., Java). The resulting application allows for modification and enforcement of network policies and actions in real time when it is loaded by the controller. The power of our application stems from its ability to use the controller APIs with high level language and all of its features. Thereby, an administrator can utilize sophisticated programming techniques to create intricate and dynamic network policy enforcement applications. The potential of such extensions is vast. For example, a network behavior learning application can levy existing techniques such as machine learning in order to more effectively police the network.

In order to provide convenient facilities for BYOD application development in the enterprise, PBS CONTROLLER consists of three high-level components: (i) *OF Channel*, (ii) *Core*, and (iii) *Admin*. *OF Channel* establishes a secure channel to PBS-DROID. The *Core* component contains Flow Manager and Policy manager to handle flows and policies. The *Admin* component includes an enterprise policy storage database, Statistics/Context Trigger to manage device information, statistics, and contexts as well as controller BYOD policy applications. Figure 9 illustrates the flow diagram and interfaces of PBS CONTROLLER. OpenFlow/PBS messages

TABLE II: Events Tracked by PBS-DROID Arbiter

Type	Description
Port Status	When applications and NICs up/down is modified.
Location	When a device enters/leaves a specific area according to the policy.
Time	When time is in a specific range according to the policy.
User Behavior	When a user changes Settings, Permissions, foreground app, etc.
Device	When the device goes into power down/up, recovery state.
Role	User's role changes (guest, employee, etc.).
Device Mode	Device switches to Normal/Sleep/AirPlane mode.

```
Target    := APP (APP_ID | APP_NAME | ALL) |
             APP_GRP (TRUST | THIRD_PARTY | UNKNOWN)
             | DEVICE (DEV_ID | GROUP |
                       UNATHORIZED | ALL)
Match     := OF_MATCH
Predicate := {Event + Condition}
Event     := PORT_STAT | LOC | TIME |
             USR_ROLE | DEV_MODE | CNTRL_STATE |
             PKT | RATE
Condition := {Operator + Value}
Value     := DECIMAL_NUMBER
Actions   := Control | Manage | Trigger
Control   := ALLOW | DENY |
             {REDIRECT | MIRROR | QUARANTINE} +
             ADDR (IP | CONTROLLER)
Manage    := REPORT | OF_ACTION
Trigger   := IMMEDIATE | PERIODIC + Value
```

Fig. 10: Syntax for High-Level Policy Language

sent by PBS-DROID are delivered to OF Channel in the controller first. It then parses standard OpenFlow and PBS messages, and passes them to the corresponding manager; i.e., the former is delivered to Flow Manager and the latter to Policy Manager. In case of new flow request messages, Flow Manager sends them to Policy Manager. Then, the messages (OpenFlow statistics, PBS Report) are passed to Statistics/Context Trigger for further monitoring, and other messages go through BYOD applications. Apart from the message flow, applications also have access to policies via the Northbound APIs. The interfaces available to administrator/app developers enable an easy way of development to handle BYOD policy enforcement.

High-level Policy & Flow Management. The Northbound APIs available to the controller-application provide an additional high-level abstract language, shown in Figure 10. This policy language facilitates the network administrative policy assignment in a simpler fashion, requiring significantly less familiarity with SDN/OpenFlow concepts in order to manage the network. The language serves as our BYOD management extension to the existing SDN controller. Our language contains four basic elements, i.e., *Target*, *Match*, *Predicate*, and *Actions*. *Target* defines the specific Android application, device, or groups which the policy targets. If APP/DEVICE is specified as ALL, it indicates the policy is applied to all applications/devices as a global property. *Match* defines a filter to match and associate a flow with a policy. *Predicate* describes a list of device context events (shown in Table II) and condition for which the policy should react, such as time or location. Lastly, *Actions* specify access control, management decision, and trigger option the policy should enforce if the predicate is satisfied, e.g., modifying a flow rule to limit device access to a network resource.

```
<Policy PolicyID=Emplyee>
 <Target app=com.facebook.android app_grp=THIRD_PARTY>
  <Match>nl_dst=66.220.144.0</Match>
  <Predicate>USR_ROLE=Business,TIME ge 0800,TIME le 1800
  </Predicate>
  <Actions>
   <Control>REDIRECT=CONTROLLER</Control>
   <Manage>REPORT</Manage>
   <Trigger>IMMEDIATE</Trigger>
  </Actions>
 </Target>
 </Policy>
```

Fig. 11: Policy Example 1

```
<Policy PolicyID=All_Unauth_Dev>
 <Target device=UNAUTH app=ALL>
  <Match>*</Match>
  <Predicate>TIME ge 0800,TIME le 1800</Predicate>
  <Actions>
   <Control>REDIRECT=123.45.67.8</Control>
   <Manage>OF_ACTION(set_vlan_id)=UNAUTH_VID</Manage>
   <Trigger>IMMEDIATE</Trigger>
  </Actions>
 </Target>
</Policy>
```

Fig. 12: Policy Example 2

Figure 11 illustrates a use case of our policy language. The policy is defined via XML and stored in the policy database. The example policy (Figure 11) implies that the administrator disallows an employee access to Facebook during work hours. Another example of a policy in practice is BYOD with NAC (Network Access Control). The enterprise may attempt a combination of the existing/outsourced NAC solution and MDM to apply traditional network policies to new BYOD devices. In such case, NAC can be used to check for the presence of an MDM agent. Endpoints that do not have the agent can be blocked or granted limited access (e.g., Internet access only). NAC can ensure that employees must comply with MDM policies if they wish to gain access to the corporate network. Building operational processes (e.g., automating mobile device registration and granting access) is key to scaling a BYOD solution. This approach can be easily implemented with our system in an incorporated way. The authentication is handled during the initial setup process (Figure 14) and network access control is automatically handled by BYOD applications developed for the enterprise. The example of the policy in Figure 12 shows a similar scenario that if a user's device is not registered, all packets are quarantined towards a middlebox via different VLAN[3]. More complex management is possible by utilizing multiple policies working in conjunction. By leveraging device context information, a static policy, or a set of policies, can

[3]Use cases in Section IV show more BYOD-specific policies in detail.

```
<Policy PolicyID=Building_Dev_Allow>
 <Target device=grp_accounting app=ALL>
  <Match>*</Match>
  <Predicate>LOC = Building A</Predicate>
  <Actions>
   <Control>ALLOW</Control>
   <Manage>REPORT</Manage>
   <Trigger>IMMEDIATE</Trigger>
  </Actions>
 </Target>
</Policy>

<Policy PolicyID=Building_Dev_Deny>
 <Target device=grp_accounting app=ALL>
  <Match>*</Match>
  <Predicate>LOC = Building B</Predicate>
  <Actions>
   <Control>DENY</Control>
   <Manage>REPORT</Manage>
   <Trigger>IMMEDIATE</Trigger>
  </Actions>
 </Target>
</Policy>
```

Fig. 13: Policy Example 3

describe device access controls dynamically, such as when the device moves from one physical location to another in the enterprise. Figure 13 illustrates this example. We specify two policies for devices which fall into a hypothetical *accounting* group. This group of devices is allowed full network access in the location of *building a*, and no network access in the location of *building b*. This example demonstrates the configuration potential and extensibility of our policy language. The policy can be easily adapted to other enterprise scenarios by administrators, with the key advantage of not requiring SDN specific domain expertise, or PBS CONTROLLER application source programming (e.g., Java).

PBS Protocol. As shown in Table I, the PBS protocol consists of four new messages: PBS_REGISTER_POLICY, PBS_REMOVE_POLICY, PBS_MODIFY_POLICY, and PBS_REPORT. The PBS protocol is designed to provide PBS-defined functions upon the existing OpenFlow protocol. Types of PBS messages are newly added as an extension. The protocol format is included in the payload of OpenFlow messages. Figure 14 depicts the sequence diagram that describes the procedure of communication via OpenFlow and PBS messages between PBS-DROID and PBS CONTROLLER. PBS-DROID and PBS CONTROLLER establish a secure connection with each other using SSL/TLS via the standard OpenFlow handshake. In this procedure, the difference from the OpenFlow handshake is that an OFPT_FEATURES_REPLY message contains mobile device information such as a unique device ID and installed app information. Namely, all the OpenFlow messages are tailored to PBS, not to the standard OpenFlow specification (i.e., traditional network-oriented specification). PBS-DROID then sends a PBS_REPORT message with user context information as shown in Table II for further authentication. Next, PBS CONTROLLER installs initial proactive flow rules and policies including global properties via OFPT_FLOWMOD and PBS_REGISTER_POLICY. Once the initial setup is completed, PBS-DROID sends an OFPT_PACKET_IN message to PBS CONTROLLER for reactive policy management. In the mean time, OpenFlow (i.e., controller-to-switch, asynchronous, and asymmetric messages)

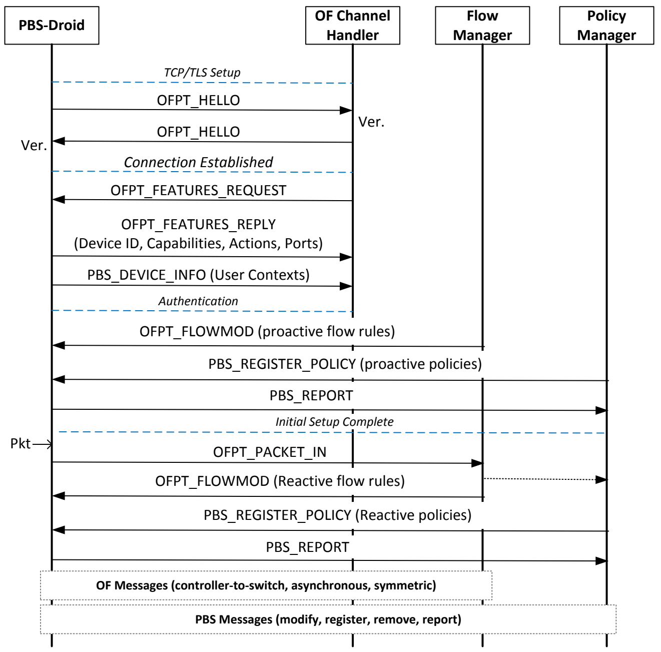

PBS-Droid
OF Channel Handler
Flow Manager
Policy Manager
TCP/TLS Setup
OFPT_HELLO
Ver.
OFPT_HELLO
Ver.
Connection Established
OFPT_FEATURES_REQUEST
OFPT_FEATURES_REPLY (Device ID, Capabilities, Actions, Ports)
PBS_DEVICE_INFO (User Contexts)
Authentication
OFPT_FLOWMOD (proactive flow rules)
PBS_REGISTER_POLICY (proactive policies)
PBS_REPORT
Initial Setup Complete
Pkt
OFPT_PACKET_IN
OFPT_FLOWMOD (Reactive flow rules)
PBS_REGISTER_POLICY (Reactive policies)
PBS_REPORT
OF Messages (controller-to-switch, asynchronous, symmetric)
PBS Messages (modify, register, remove, report)

Fig. 14: Communication via PBS Protocols

and PBS messages (i.e., register, modify, remove, and report messages) can be sent to the controller simultaneously for further management.

F. Optimization Schemes

A significant challenge in Android devices is limited resources. In this section, we discuss how PBS-DROID minimizes the overhead while running on mobile devices.

Message PushDown. Due to our empirical experience with SDN, we found that the message handling overhead (e.g., processing frequent new-flow requests and periodic statistical messages) greatly throttles the performance of both a network device and the controller. A simple mitigation for the overhead can be addressed by controller applications by increasing the flow timeout and statistics request period. However, a high timeout period can cause the device flow table quickly occupied to the limited size. A typical limitation in SDN requires the application or controller to choose between a time trade-off in flow rule persistence in the table by adjusting the request period to an optimal request period. We mitigate this trade-off by utilizing the policy table to specify a more detailed flow timeout value which alleviates the burden of specifying a trade-off optimal request period. For example, the device context such as location or time can automatically remove a rule when it is no longer needed, thus we do not need to guess how long such a rule is relevant in the device table.

Two-tiered Programming Model. Although the message pushdown scheme reduces the burden of handling new flow requests using device context information, the administrator may require detailed, active monitoring of device network information. This entails sending flow information to the controller in real time, undermining our previous pushdown optimization. To mitigate this, we utilize minimum intelligence to the device, which can react/program upon device events in real-time to efficiently reduce communication overhead and save processing overhead in the controller. Different from the existing SDN switch software which has no conditional logic, PBS-DROID maintains policy engine in the management plane, where the policy table is managed. The policy language, as shown in Figure 10, enables the administrator to define complex predicates/conditions varying from user context to packet statistics in conjunctive forms. Also, our *actions* support OpenFlow actions and flow modification schemes as well as BYOD security-related actions. Thereby, with low interaction with PBS CONTROLLER the flow can be efficiently managed inside the device.

Tailor to Mobile Device. When it comes to the mobile device environment, simple adoption of existing SDN software does not fit well on the device. This is because the SDN specification targets traditional network devices and operations, not the mobile device environment. For example, periodic operations such as port monitoring, reconfiguration checking, and LLDP (Link Layer Discovery Protocol), specified by OpenFlow, can adversely affect the battery life and bandwidth available to mobile devices, which is not a concern in traditional network switches. However, many of the periodic operations (e.g., reconfiguration checking, LLDP) do not impact the basic functionality of PBS-DROID. Rather, those operations are not required for BYOD contexts. Also, operations dedicated merely to traditional networks can be ruled out or tailored to the mobile device. For instance, emergent flow and fail-open/over mode are not required for PBS. The operation of rate limit is redesigned to work as a conditional rate limit by policy. Thus, we do not need the full set of the specification functionalities. We remove such periodic operations and facilitate an event-driven approach to improve PBS-DROID performance. For instance, the userspace daemon does not check port status and statistics in a *while* loop; rather, the status checking is notified in an event-driven fashion through a *netlink* from the Arbiter and a statistics message is sent to the controller only when a policy action is designated as *Report*.

Short-circuit Operation. A performance bottleneck of PBS-DROID is drawn from the packet processing pipeline. The pipeline processes a packet from an app as such: (i) a mapping of application ID to a virtual interface, (ii) redirection of the packet to the virtual interface, (iii) a flow table lookup (further *sk_buff* copy and communication overhead in case of miss), and (iv) a policy lookup. As defined in Figure 10, our policy language allows the administrator to define global properties that can apply to all devices or all applications. We note that pipeline performance can be optimized by checking global properties early in the pipeline. The short-circuit operation thus checks for such bulk cases, allowing a skip of subsequent pipeline steps where appropriate. This optimizes the performance and saves CPU cycles, which is important to the mobile device battery.

G. Implementation

Our prototype implementation of PBS CONTROLLER is built as an extension to Floodlight v1.0. The extension runs as Floodlight modules without impacting existing Floodlight components and provides northbound APIs to BYOD controller-applications we author. Although we currently only support Floodlight, it is not difficult to extend PBS CONTROLLER to other SDN controllers, which we aim to address in future work. PBS-DROID currently supports OpenFlow 1.0 and above for a secure communication between PBS-DROID and PBS CONTROLLER. This can also be extended in the future through the use of additional controller implementations. Our prototype implementation of PBS-DROID leverages an existing software switch, *Openvswitch*, which consists of three core components: (i) *ovs-vswitchd*, (ii) *datapath*, and (iii) *ovsdb*. *ovs-vswitchd* is a userspace daemon that establishes a secure communication channel to the controller and intercommunicates with the kernel datapath. The userspace datapath maintains the full flow table and the kernel datapath keeps a cache for microflow and megaflow for lookup speedup. *ovsdb* maintains a persistent local database to store network configurations (i.e., bridge, ports, etc.). Beyond this architecture, we modified the architecture with our optimization schemes dedicated to mobile devices and added new components designed in Section III. For the new components, we implemented: (i) application abstraction layer in the framework layer to provide application awareness, (ii) PBS protocol handler in our secure control channel, (iii) PBS-DROID Data Plane and Management Plane in the kernel space, and (iv) Arbiter to manage various user/device contexts at run-time.

IV. EVALUATION

Our design goals are fine-grained access control, dynamic policy enforcement, network-wide programmability, and minimal performance overhead impact. In accordance with these goals, our evaluation is two-fold: (1) to measure its performance overhead (network level and system level) and (2) to demonstrate use cases which showcase our capabilities to enforce security policies on mobile devices.

A. Performance Overhead

Testing Environment. To measure the performance overhead imposed by PBS-DROID on a real Android Device, we use an LG Nexus 5 equipped with a Qualcomm MSM8974 Snapdragon 800 CPU and an Asus Nexus 7 tablet with an ARM Cortex-A9. Both testing devices run Android system version 4.4 (KitKat). Our network controller runs on Ubuntu Linux x64 with a Quad Core CPU with 8 GB of memory.

1) Network Performance Overhead: The network overhead of PBS is mainly incurred by the flow installation and policy enforcement processes. These processes are always accompanied by a flow lookup, a new flow request, and a flow/policy installation between the device/controller over the network. Thus, it is imperative that the network performance evaluation should include flow and policy processing overhead. We measure this processing delay by testing the average round trip time of 100 packets from our testing devices (with and without PBS) to popular high availability servers (e.g., google.com). As shown in Table III, there is no notable increase in packet round trip time introduced by PBS. We also note that these delays include the flow installation overhead for the first unknown incoming packet which is much higher than the processing time for the following known packets. Once the flow rule is installed all packets in the same flow will no longer need to go through the controller. Both devices with PBS show

Server	NX5	NX5(PBS)	NX7	NX7(PBS)
google.com	16ms	18ms	18ms	20ms
facebook.com	43ms	47ms	45ms	49ms
yahoo.com	59ms	64ms	40ms	42ms

TABLE III: Average Packet Processing Overhead

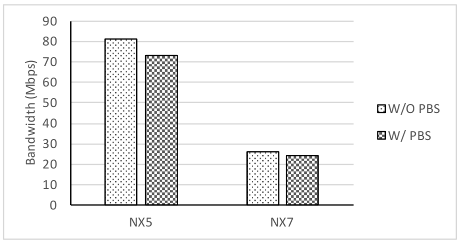


Fig. 15: Throughput Benchmark

similar packet processing delay. The overhead incurred by PBS is negligible for both devices, inline with our goal.

In addition to packet processing overhead measurement, we also test the overhead for PBS system on network throughput by using a standard tool, *iperf*. In particular, we open the *iperf* server as a communication port to listen to the client application on the device, and set test duration as 10 minutes with a two-second interval between periodic bandwidth reports. As shown in Figure 15, the average overhead of bandwidth for the Nexus 5 shows approximately 9% overhead and 7% for the Nexus 7. Our evaluation results demonstrate that the network bandwidth overhead is acceptable. We attribute this performance impact to the implementation of the flow cache and policy tables in the operating system kernel space.

2) System Performance Overhead: We evaluate the system performance in the following order. (1) We compare the overall system performance scores with and without PBS as measured by standard Android benchmark tools. (2) We break down and compare the overall scores into individual performance metrics: CPU, memory, and battery overhead.

Benchmark Tools. We use four representative Android benchmark tools for the evaluation: *Antutu*, *Geekbench*, *Vellamo*, and *PCMark*. *Antutu* is one of the most popular benchmark tools and comprehensively tests all aspects of the device, including CPU, RAM, I/O, etc. *Geekbench* provides a comprehensive measurement *relative* to other Android devices in a scaled cumulative-score-manner. *Vellamo*[4] encompasses several system-level performance metrics such as I/O, mobile processor, and browsing. *PCMark*[5]'s tests are based on everyday activities and reflect real-world performance by measuring the battery life of the device.

Overall System Performance Overhead. In Figure 16a and 17a, we show the overall benchmark scores. A higher score denotes better performance in each scale/category. Based on our evaluation, we observe an overall overhead of 5.3% in *Antutu*, 1.1% in *Vellamo*, 6.0% in *GeekBench*, and 5.7% in *PCMark*, introduced by PBS-DROID on the Nexus 5 as depicted in Figure 16a. On the Nexus 7, the overheads are 1.4% in *Antutu*, 5.3% in *Vellamo*, 5.0% in *Geekbench*, and 2.2% in *PCMark* respectively as shown in Figure 17a. The slight differences between two Nexus platforms are mainly due to the different hardware specifications and software/application environments. Overall, the overheads for both devices are in a reasonable range, no larger than 6% for either devices. Thus, PBS-DROID does not exhibit significant system overhead from our benchmark results.

[4]We use two metirics, *Multicore* and *Metal*, from the *Vellamo* benchmark.
[5]We use all metrics of *Work Performance* (*Browsing*, *Writing*, *Video Playback*, and *Photo Editing*).

Attribute	NX5	NX5 PBS	Over.	NX7	NX7 PBS	Over.
Browsing	3741	3926	4.95%	1972	2176	10.34%
Writing	3174	3436	8.25%	2202	2301	4.50%
Video	4118	4276	3.84%	3739	3893	4.12%
PhotoEdit	4804	4948	2.99%	2591	2597	0.23%
Total	15837	16586	4.73%	10504	10967	4.41%

TABLE IV: Battery Overhead Measurement (lower is better)

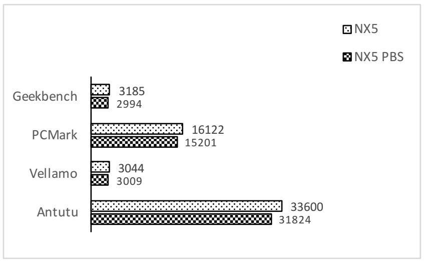
CPU and RAM Overhead. To evaluate the CPU overhead introduced by PBS-DROID, we run two benchmarking tools, *Geekbench* and *Vellamo*, which include corresponding metrics in comparison. As shown in Figure 16b, *Geekbench* shows an overhead of 5.9% and *Vellamo* shows 2.7% overhead against the without-PBS case on the Nexus 5. Also, we observe 5.3% overhead in *Geekbench* and 7.3% in *Vellamo* on the Nexus 7 as shown in Figure 17b. In addition, we run two benchmarks, *Geekbench* and *Antutu*, to evaluate RAM overhead. We observe 4.2% overhead in *Antutu* and 4.8% overhead in *Geekbench* for the Nexus 5 and 0.8% overhead in *Antutu* and 0.2% overhead in *Geekbench* for the Nexus 7 as shown in Figure 16c and 17c respectively. The results for both devices showed negligible overhead on CPU and RAM evaluation. Overall, we conclude that PBS-DROID poses a reasonably low impact on CPU/RAM overhead.

Battery Overhead. In order to verify our goal, we evaluate the battery overhead to show its feasibility on a resource-limited mobile device. We use *PCMark* for the evaluation. Before testing, we charge the battery to full capacity (100%) and test until the charge drops below 20%. Table IV summarizes the overhead by battery attribute from *work performance*: browsing, writing, video playback, and photo editing. The writing metric shows the highest overhead of 8.25%, browsing attribute incurs 4.95%, and the rest of evaluation attributes show less than or around 4% on the Nexus 5. Additionally, we note that PBS shows the highest overhead of 10.34% for Browsing, while all other metrics show less than or around 4% overhead on the Nexus 7. Overall, we note an average of 4.73% battery overhead on the Nexus 5 and 4.41% on the Nexus 7 when using PBS-DROID. Accordingly, the evaluation shows that PBS-DROID does not incur significantly greater battery consumption compared to the stock Android device.

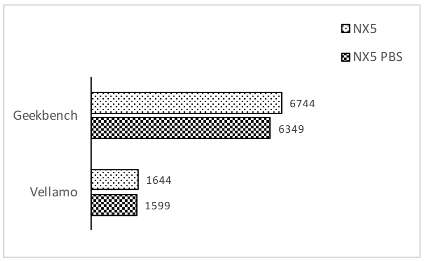
B. Use Cases

In this section, we demonstrate that security policies can be effectively enforced with PBS.

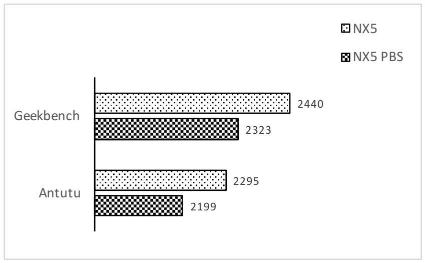
Deployment Environment. In the scope of our evaluation, we deploy PBS-DROID to manage three buildings in our campus facilities. Figure 18 shows the PBS-DROID enabled scenario, designating the buildings as (B_a, B_b, B_c) which together form our central controller-managed network policy deployment. The colored areas of Figure 18 designate the local-based enforcement radius of each building. Devices



(a) Benchmark Result

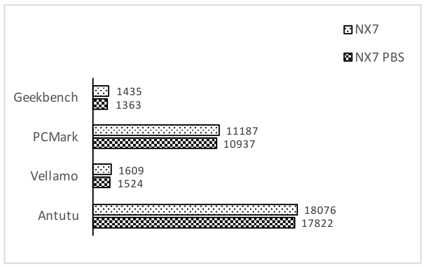


(b) CPU Overhead

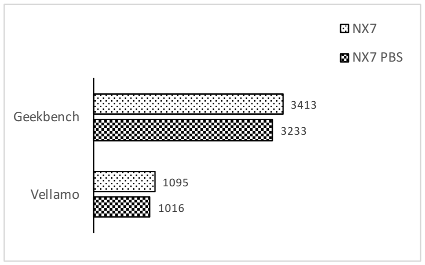


(c) RAM Overhead

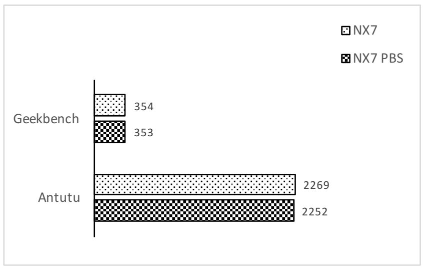
Fig. 16: PBS System Performance Evaluation (Nexus 5)



(a) Benchmark Result



(b) CPU Overhead



(c) RAM Overhead

Fig. 17: PBS System Performance Evaluation (Nexus 7)

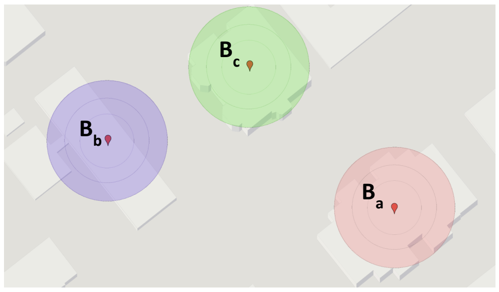


Fig. 18: Managed Facilities

within the radius will fall under location-specific context policies, as measured by on-device high-accuracy GPS. Note that the additional elevation sensitive radius is not shown in the figure, which forms a location sphere encompassing the entire building roughly.

Admittedly, building dimensions are seldom regular, making it difficult to achieve perfect coverage. We note that a more fine-tuned area specification with multiple overlapping, smaller coverage areas can provide a higher resolution coverage of building volume. However, since we seek to demonstrate our system capability in a simple and concise fashion, we utilize a basic building coverage designation for the sake of clarity.

Each building provides independent network connectivity via multiple short range wireless 802.11g access points (APs). In order to ensure device control, connectivity between the PBS-DROID managed devices and the controller is maintained throughout the devices life cycle in the network.

1) Use Case 1: Network Activity Logging: Network logging and measurement is an essential feature in network administration. The ability to monitor and log detailed network connection information is extensively useful in network debugging, configuration validation, and security audits.

We evaluate the capability of PBS-DROID to easily facilitate device network activity logging, by developing a demonstration controller application, called *netlog*. While simple in construction, less than 300 lines of Java code, *netlog* facilitates both network view in depth and breadth. We provide the administrator a view of network-wide information, participating device coverage across multiple buildings, subnetworks and locations, while we simultaneously enable deep, device-application specific connection information and network utilization statistics in real time.

Our application logs our testing environment device connection activity, network history, and location updates. The log data provides an insight into detailed device activity on the network previously invisible, such as the device-application specific connection information. For example, we can observe the time and location a device invoked specific applications (email, social, enterprise specific, etc.). Furthermore, we utilize a simple filter which narrows our gaze to individual devices and provides an invaluable tool in formulating and fine-tuning detailed network policies.

2) Use Case 2: Network Policy Enforcement: Although network logging is essential to the passive administrator, it is of little value without realtime context-specific programmability to act upon it. The key insight is, PBS-DROID is capable of reacting based on the device's network and context specific information.

We assess the ability to provide such capability by defining a dynamic network policy to be enforced by our prototype SDN controller application, *netpol*. Our policies demonstrate useful and contextual device specific attributes such as *Appli-*

cation Specific Policy, *Device Context Policy*, and *Device Role & Authorization*.

In our test case, we use two network mobile devices $(D1, D2)$, across multiple buildings described earlier. The devices run several common android applications, $(Dn_{a,b,c,d})$. Taking on the administrative role, we specify a simple network policy designed to control communication between two devices on the network based on device, application, and time context. The policy aims to restrict inner network device and host connections, specified in Table V.

Policy Action	Flow	Application	Context
Deny	D1 to/fro D2	a	is 08:00-12:00
Deny	D1 to/fro D2	a	is 12:00-17:00
Allow	D1 to/fro D2	a	is 12:00-13:00
Deny	D1 to H1	b	in B_a
Deny	D2 to H1	b	in B_a

TABLE V: *Policies specifying application, device, and time restrictions by context.*

When running application a, all network connection attempts are monitored by the central network controller provided by PBS-DROID. When an instance of a attempts to establish a connection between either device, the policy enforcement is triggered via a flow table lookup. If no rule exists, the controller is queried for a policy decision. In the case of the time context, the controller will issue a proactive rule update precisely at noon, updating the policy table on each device. We may also restrict application access to specific hosts or services based on their location. For example, application b may not access host $H1$, a secure database, unless the device is present in the designated building.

An alternative policy scenario specifies location-based connection management based on device role. We designate device $D1$ as a trusted supervisor while $D2$ remains in a standard employee role.

Connection Policy	Device Location	Application
Allow	B_b	$D1_a$
Deny	B_b	$D2_a$
Allow	all	c
Deny	all	b

TABLE VI: *Location and Role Restriction*

When the device context switches from one location to another the controller (app) is notified, which in turn triggers a local policy modification in the device flow table (as indicated in Figure 19). In our scenario (as specified in Table VI), application a has context-specific access based on the device utilizing the app. While allowed by the supervisor $D1$ in building B_b, an authorized device $D2$ may not run a in B_b. Wildcard policies may also be applied. For example, we never allow application b to access the network regardless of its location, while an enterprise-authorized application, c, is granted all accesses.

3) Use Case 3: Application Flow Path Management: In addition to simple blocking based policies we also evaluate more sophisticated actions, such as traffic redirection for the purpose of security and network load management.

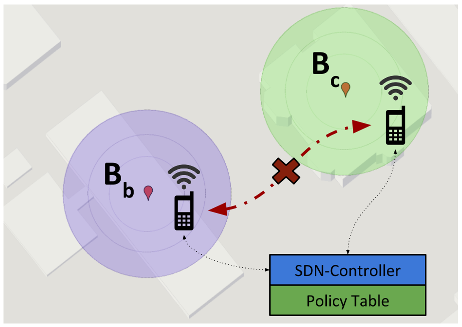


Fig. 19: Inner Network Communication Restriction

We demonstrate such capability at device-application level by implementing a load balancer controller-application, called *netbal*. In addition to supporting device traffic routing, we also leverage our ability to differentiate among application-specific data flows. In our scenario, we attempt to balance the bandwidth utilization across two applications a and b, regardless of their host devices. Because we have access to network connection statistics at run time, *netbal* can make the decision to reroute all traffic originating from application a to a different network end point. Another interesting application of BYOD-enabled facilities is to use existing, dedicated, security middle-boxes to inspect suspicious device/app activity when necessary. To do so, we modify *netbal* to facilitate a different type of network rerouting, i.e., quarantine redirection. We create a basic white-list of authorized applications (a,b,c) and redirect all other application traffic to a middle-box running Snort for inspection.

V. LIMITATIONS AND DISCUSSION

Although PBS is able to provide dynamic fine-grained policy enforcement with low overhead impact in BYOD devices, there still remain hurdles in our current prototype implementation. Here, we discuss those limitations and our future work.

System Circumvention. Some malicious user activities may intentionally seek company policy circumvention/violation. For example, in order to gain access to otherwise restricted services/accesses, a user could attempt to simply turn off their context function such as GPS. PBS-DROID, however, is aware of such context switch updates, in which case the device can be quarantined until GPS function is restored. Furthermore, although our system drops all packets in case of loss of secure connection, a user may attempt to turn off our system by killing its associated processes and the kernel module. However, this requires a user to obtain root privilege on the device. Furthermore, a user may attempt to ignore company policies by utilizing a device with no PBS-DROID installation or an existing work device tampered to remove PBS-DROID. We note that our system can operate in conjunction with existing complementary *MDM* solutions. The network administrators can rely on many available MDM systems [5], [8], [4] for the assurance that a device used in the workplace is properly authenticated, equipped with the proper software client, and vetted for operation in the network, a common BYOD enterprise practice [7].

Portability. Portability is an important factor in achieving the necessary coverage in a BYOD deployment, which may be composed of a multitude of operating system versions and mobile devices. In order to support multiple versions of Android, our system requires a few minor modifications. Unavoidably, tightly integrated Android components, such as the Arbiter, may require some source code modification to support system/device specific information (e.g., PackageManager, GPS, etc.). While PBS-DROID is an Android specific implementation, our underlying design principle is portable to other platforms. PBS-DROID can be extended to Linux, Windows and Apple devices/machines with additional engineering efforts. We acknowledge that not all BYOD devices may support the full range of our context-based policies, e.g., devices that lack GPS capability will be limited in location-based policy enforcement. In such a case our system may utilize a more rudimentary location based check, e.g., grasping information from the physical network connection of the device, such as wireless access point or subnet information.

Protocol and Interface Coverage. First, our current implementation of PBS-DROID only covers TCP network flows. While many Android applications utilize standard TCP network communication, it is possible to circumvent our prototype via other protocols such as UDP. We note that SDN/OpenFlow does support both TCP and UDP, as well as ICMP. Extending our system to provide more comprehensive protocol coverage is not difficult and merely needs more engineering effort, which is a focus of our immediate future work. Second, as noted in our design section we support both WiFi and 3G/4G interfaces on the mobile device. There are, however, other interfaces which an application may utilize for communication such as Bluetooth and NFC (near field communication). Our intention is to achieve greater interface coverage in future work, which couples with broader protocol support as not all interfaces may utilize the TCP/IP network stack.

Scalability of Controller. We assume that a BYOD application is working on a normal-size enterprise network since the BYOD application running on top of a controller is treated just as a normal SDN controller application. Our components to handle enterprise policies on top of flow management incur not much performance overhead compared to normal SDN applications running on a single controller. We note that supporting large-scale enterprises is still an important matter, which attributes to the scalability performance of SDN controllers/networks. Existing research [14], [10], [28] on SDN already shows promising results in reasonably large enterprise/cloud/data-center networks.

SDN Attacks. Since PBS-DROID utilizes SDN concepts, this might make it vulnerable to existing SDN attacks, such as control plane saturation, which denies controller availability [24]. An administrator, however, could mitigate such a threat through the use of network policies which limit device-specific network traffic or temporarily block offending devices. Existing SDN security systems [22], [24], [20], [13], [11] can aid in the protection of a PBS managed network. Thus, security concerns on SDN-centric vulnerabilities are not the focus of this paper.

VI. RELATED WORK

In this section, we review several relevant existing work.

Android Systems at Enterprise. Android Device Administration provides an API [2] which facilitates the creation of stronger, security-aware Android applications. The API provides a succinct number of static security functions, such as limiting the minimum on device password length, disabling the camera, and imposing shorter inactivity lock-out timers. The static policies, however, fall short in addressing aforementioned issues, lacking the ability to provide reactive, or real-time programmable policies or features. Android for Work [1] and Samsung KNOX [6] provide a dedicated profile/container system which separates corporate data from personal data on the users' devices. These systems expose APIs to MDM solutions, which combined present a unified management framework/platform for IT admins. Yet their feature set is limited in functionalities for dynamic, fine-grained network policy management, which is the focus of this work.

Some existing work [12], [9], [18], [27], [17] provides device access restriction via policy enforcement. Operating at kernel and application layer instead of network level, however, these solutions do not provide access control to device-external resources such as remote host communication. Despite using policies, the above solutions do not support dynamic programmability, limiting their functionality to only passive static policy enforcement. AirBag [26] uses a virtualization-based approach to achieve isolation of trusted applications. This solution can be extended to isolate personal apps from those managed by the enterprise. Such approach is complementary to PBS-DROID. Another recent work, DeepDroid [25], also addresses the problem of BYOD policy enforcement in the enterprise. By tracking the system APIs (system_server, zygote, binder transactions), DeepDroid can enforce app-context-aware policies. However, DeepDroid is limited by static policy configuration, much like other previous work. Without the ability to offer a programmable management interface, DeepDroid cannot account for scenarios which require reactive network action, such as application-specific load balancing, quarantine and context switch response in real time.

Lastly, meSDN [15] is another work which adopts the concepts of SDN in the Android environment. The authors solve a different problem where they use SDN in order to optimize interaction between cloud infrastructure and wirelessly connected devices. They focus on WLAN virtualization by moving app logic into the central controllers. Our solution is broader in that we consider all network interfaces and data flowing through each of these interfaces. We also provide context-aware policies and network access control in PBS-DROID, which is not provided at all by meSDN.

SDN/OpenFlow Security. Insufficient security considerations in SDN/OpenFlow has led to an increased focus in recent research. FRESCO [22] presents an OpenFlow security application development framework which provides modular composable security services for application developers. Avant-Guard [24] provides connection migration techniques to solve the challenge of scalable control plane saturation attacks. TopoGuard [13] is designed to solve new topology poisoning vulnerabilities/attacks. SPHINX [11] provides a flow-graph model to detect various traffic flow related attacks in SDN.

FortNox [20]/SE-Floodlight [19] support role-based authorization and security constraint enforcement in order to solve flow rule contradictions in real time, as well as providing several other security protection at the control plane. Rosemary [23] is a new security-oriented SDN controller which strengthens the control plane. Different from all existing work, our paper focuses on providing a network security policy enforcement system for BYOD devices in enterprise networks.

VII. Conclusion

In this paper, we propose a new network security framework for BYOD in enterprise networks, i.e., PBS (Programmable BYOD Security). Motivated by the new concept of Software-Defined Networking (SDN), we provide an application and network mobile devices management and policy enforcement system PBS-Droid on the Android platform. We achieve dynamic, fine-grained network control of applications on mobile devices. With PBS, administrators also benefit from the global network visibility and fine-grained policy programmability. We introduce PBS-Droid as a concrete client implementation for Android platforms. Without imposing much performance overhead, PBS-Droid can effectively enforce the dynamic network access control policy with consideration of users' context information. We believe PBS greatly complements existing security solutions and represents a new direction for the important BYOD security domain.

Acknowledgment

This material is based upon work supported in part by the Air Force Office of Scientific Research under FA-9550-13-1-0077 and a Google Faculty Research award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFOSR and Google.

References

[1] Android for Work. https://www.google.com/work/android/.

[2] Device Administration. http://developer.android.com/guide/topics/admin/device-admin.html.

[3] MDM Comparison. http://www.tomsitpro.com/articles/mdm-vendor-comparison,2-681.html.

[4] Miradore. http://www.miradore.com/miradore-online/.

[5] Mobile Device Management. http://en.wikipedia.org/wiki/Mobiledevicemanagement.

[6] Samsung Enterprise. http://www.samsung.com/us/business/samsung-for-enterprise/index.html.

[7] Samsung KNOX. http://www.samsung.com/global/business/mobile/platform/mobile-platform/knox/.

[8] SPICEWORKS. http://www.spiceworks.com/free-mobile-device-management-mdm-software/.

[9] K. Z. Chen, N. Johnson, V. D'Silva, S. Dai, K. MacNamara, T. Magrino, E. X. Wu, M. Rinard, and D. Song. Contextual policy enforcement in android applications with permission event graphs. In *NDSS*, 2013.

[10] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. In *Proceedings of ACM SIGCOMM 2011 Conference*, August 2011.

[11] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. Sphinx: Detecting security attacks in software-defined networks. In *In proceedings of the 22th Annual Network & Distributed System Security Conference (NDSS'15)*, 2015.

[12] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245. ACM, 2009.

[13] S. Hong, L. Xu, and H. Wang ang G. Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *In proceedings of the 22th Annual Network & Distributed System Security Conference (NDSS'15)*, 2015.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[15] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K. Kim, and T. Nadeem. mesdn: Mobile extension of sdn. In *Proceedings of the Fifth International Workshop on Mobile Cloud Computing &#38; Services*, MCS '14, pages 7–14, New York, NY, USA, 2014. ACM.

[16] Logicalis. BYOD ? Research findings. http://cxounplugged.com/2012/11/ovum_byod_research-findings-released/.

[17] M. Nauman, S. Khan, M. Alam, and X. Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. ACM, 2010.

[18] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. *Security and Communication Networks*, 5(6):658–673, 2012.

[19] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran. Securing the software-defined network control layer. In *In proceedings of the 22th Annual Network & Distributed System Security Conference (NDSS'15)*, 2015.

[20] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'12)*, August 2012.

[21] S. Shin and G. Gu. CloudWatcher: Network Security Monitoring Using OpenFlow in Dynamic Cloud Networks (or: How to Provide Security Monitoring as a Service in Clouds?). In *Proceedings of the 7th Workshop on Secure Network Protocols (NPSec' 12), co-located with IEEE ICNP' 12*, October 2012.

[22] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS'13)*, February 2013.

[23] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 21th ACM Conference on Computer and Communications Security (CCS)*, 2014.

[24] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[25] X. Wang, K. Sun, Y. Wang, and J. Jing. Deepdroid: Dynamically enforcing enterprise policy on android devices. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[26] C. Wu, Y. Zhou, K. Patel, Z. Liang, and X. Jiang. Airbag: Boosting smartphone resistance to malware infection. In *Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS'14)*, 2014.

[27] R. Xu, H. Saïdi, and R. Anderson. Aurasium: Practical policy enforcement for android applications. In *USENIX Security Symposium*, pages 539–552, 2012.

[28] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable Flow-Based Networking with DIFANE. In *Proceedings of ACM SIGCOMM 2010 Conference*, August 2010.