Per-user Policy Enforcement on Mobile Apps through Network Functions Virtualization

Amedeo Sapio
Politecnico di Torino
Torino, Italy

Yong Liao
Narus, Inc.
Sunnyvale, CA

Mario Baldi
Narus, Inc.
Politecnico di Torino

Gyan Ranjan
Narus, Inc.
Sunnyvale, CA

Fulvio Risso
Politecnico di Torino
Torino, Italy

Alok Tongaonkar
Narus, Inc.
Sunnyvale, CA

ABSTRACT

Due to the increasing popularity of smartphones and tablets, mobile apps are becoming the preferred portals for users to access various network services in both residential and enterprise environments. Predominantly using generic HTTP or HTTPS protocols, traffic from different mobile apps is largely indistinguishable. This loss of visibility into mobile app traffic brings new challenges to network management and traffic analysis. It has became very hard to implement network policies based on the differentiation between traffic from compliant and non-compliant mobile apps. This paper presents a system that not only provides network administrators the much desired capability of enforcing policies on mobile app traffic, but also does that at a fine *per-user* granularity. The proposed system takes a *Network Functions Virtualization (NFV)* approach and virtualizes an edge router into multiple virtual data planes. Specifically, each data plane serves solely to one particular user and consists of user-specific virtualized network functions. The independence of the virtual data planes facilitates enforcing network policies at the per-user level. To enable policy enforcement on mobile apps, our system includes a sophisticated mobile app identification module to recognize traffic from different apps using preloaded traffic signatures. By exploiting TLS proxying, our system can even enforce policies on those mobile apps adopting traffic encryption. We have implemented a prototype of the proposed system as a wireless access point (AP) using a commodity small form factor PC. Our preliminary experimental evaluations show that the system can scale to modest number of users without much impacting user experience in using the network.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network communications*

Keywords

Network Functions Virtualization; Mobile Security; Policy Enforcement; Traffic Filtering

1. INTRODUCTION

State-of-the-art firewall and network monitoring systems rely largely on IP address, domain-name, and port number based policy formulation and enforcement [6]. Such approaches are becoming untenable as user-owned devices (mostly smartphones and tablets) are increasingly accepted within enterprise networks (*a.k.a.* the bring-your-own-device (BYOD) to work phenomenon). Such devices execute a multitude of mobile applications (roughly 1.75 million at the last count of Android and iOS markets), a significant portion of which use HTTP and HTTPS to interact with servers connected on Internet. To make matters worse, such servers are often times hosted by cloud providers or content delivery networks. Thus, conventional protocol and port number based traffic analysis tools can neither differentiate traffic from distinct mobile apps, nor separate mobile apps traffic from generic *web-traffic* [4, 14]. The advent of Web 2.0, which facilitates rapid development of web-based and distributed applications, has only accentuated the problem as many enterprise and consumer web applications use the same protocols.

While some mobile applications may actually be necessary or useful in corporate or residential context, others may compromise information security of a network [13]. Besides the issue of different applications, the same application developed for different platforms (e.g. Android vs. iOS) may have different security vulnerabilities. Finally, as mobile applications increasingly become multi-faceted and complex, the risks of them being used for data exfiltration have escalated [9, 7]. For example, popular services such as Facebook now functions as authentication gateways and substrates for a large eco-system of applications (e.g. FarmVille). This can provide indirect access to sensitive information to undesired, and potentially malicious, third parties.

Additionally, roles and privileges within a network might need to be different on a per-user basis. For instance, in an enterprise while certain employees might need to access and/or share sensitive information with prospective clients for business objectives, others should not be permitted to do so. Similarly, in a residential network environment, parents may want to restrict their kids' access to applications that are not child-safe.

Addressing these challenges calls for a new breed of policy formulation and enforcement systems based on a completely new design. To guarantee network and information security, it is impera-

tive that such systems have the capability to identify and differentiate traffic generated by different mobile applications, across platforms and devices, and impose policies based on users and their roles.

In this paper we propose the MAPPER *(Mobile Application Personal Policy Enforcement Router)* system, which can implement per-user fine-grained policies on users' mobile app usage by monitoring network traffic. The presented MAPPER implementation is a wireless network access point that, upon authenticating a connecting user, loads a set of modules to process network traffic to/from the user's device, and implements user-specific access policies based on the user applications that generated the traffic. The design of MAPPER is inspired by research in the field of *Network Functions Virtualization (NFV)*. More specifically, MAPPER allocates each user a dedicated virtual execution environment to run network functions specialized to the user. The identification of mobile apps generating traffic flows is provided by a sophisticated engine integrated into MAPPER as a virtualized network function. This engine processes HTTP flows and identifies rich information on the origins of those flows, such as the mobile apps that generated those flows and the platforms (Android, iOS, or Blackberry) of those apps. To maximize the visibility into network traffic, MAPPER can additionally instantiate a virtualized network function running a TLS-capable man-in-the-middle proxy, which transparently terminates HTTPS connections so as to retrieve plain-text data from encrypted flows.

NFV plays an essential role in the design of MAPPER. By Isolating network functions operating on the traffic of each user basis, it greatly eases the enforcement of per-user policies and separation of traffic from different users, which is key from both privacy and performance perspectives. Network functions, as well as the policies they implement, can migrate seamlessly across different MAPPER access nodes, in order to follow the user and enforce the policy no matter which network access node she is connected (e.g., independently of the department or branch of her company she is visiting), providing a uniform security protection.

The design and development of MAPPER offers a number of technical contributions presented in this paper. To the best of our knowledge, MAPPER is the first system capable of enforcing rich policies on mobile app traffic on a per-user basis. The NFV approach taken in MAPPER provides extreme flexibility in architecting the system and facilitates some unique features of the system. We have implemented a prototype of MAPPER to demonstrate its feasibility. Our tests show that the ambitious goal of enforcing policies on a vast number of mobile apps at the fine per-user granularity, even for apps encrypting their traffic, can be achieved with off-the-shelf commodity hardware.

The rest of this paper is organized as follows. Section 2 surveys the related works on remote policy enforcement, mobile app identification, and network functions virtualization. Section 3 presents the design of MAPPER.Performance evaluation results are presented in Section 4. Section 5 concludes this paper and discusses future steps to improve the system.

2. RELATED WORK

The state-of-the-art remote policy enforcement schemes exploit dedicated middleboxes that inspect network traffic with certain degree of application level awareness (e.g. firewall) [6]. Approaches based on Virtual Private Networks (VPN) leveraging Trusted Platform Module (TPM) chips are proposed in literature as well [12]. These solutions are no longer suitable for networks serving mostly mobile devices. First, those mobile devices use a new communication paradigm, mobile apps, that works mainly on HTTP/HTTPS protocol, and the state-of-the-art approaches do not have the deep and fine-grained visibility into HTTP/HTTPS traffic. Secondly, it is often hard for enterprise network administrators to get control of the devices directly due to the lack of some software/hardware features in those devices, and the devices are usually not owned by enterprises (due to the BYOD trend).

The capability of deeply analyzing network traffic to identify which mobile apps have generated the traffic is studied in a series of recent works [15, 4, 14]. The richness of app identifier information embedded in advertisement traffic of mobile apps is studied in [14]. Dynamically executing mobile apps in sandbox environment so as to generate state machine based mobile app traffic signatures is studied in [4]. The FLOWR system presented in [15] starts from a small set of seeding knowledge to automatically learn more mobile app signatures by observing large volume of network traffic. MAPPER integrates those research results into its mobile app identification engine.

The basic architecture of MAPPER is inspired by the idea of Network Function Virtualization, which has long been an active research field. The virtualization on commodity network hardware has been proven to be a powerful and practical proposition by many works, such as Flowstream [5], RouteFlow [10], and NetVM [8]. We chose to base our work on the FROG system [11, 2], because it offers a powerful and extensible architecture to separate the traffic on per-user basis by assigning each user a dedicated lightweight virtual machine. This capability was not provided and analyzed by the other above mentioned works.

3. SYSTEM ARCHITECTURE

3.1 Overview

Leveraging the FROG system, MAPPER allocates isolated virtual environments to execute network functions. A network function executed in a MAPPER virtual environment is referred to as a *net-app*. Each user connected to a MAPPER device is assigned with a dedicated virtual environment to execute net-apps specific to the user. Such a per-user virtual environment is referred to as *Personal Execution Environment (PEX)*. A PEX is like an edge router, with the capability to run arbitrary net-apps, designated to process traffic of only one user. MAPPER also has a second type of virtual environment for executing net-apps that are not specific to any given user. Such a virtual environment is referred to as a *Global Execution Environment (GEX)*. A GEX can run services that (*i*) are not required to be executed in a per-user fashion and (*ii*) may be needed by multiple users. In that case, a GEX provides a certain *service* that is exposed via a pre-defined API. A net-app running in a user's PEX can access the service by calling its API.

The capability to distinguish traffic from different mobile apps is provided by the *Mobile Application Identification (MAI)* net-app running in a GEX. MAI runs a sophisticated engine to process HTTP flows and identify mobile apps that generated them. Visibility into encrypted traffic, such as HTTPS flows, is provided by a TLS-capable *man-in-the-middle proxy (MiMP)* net-app running in another GEX, which transparently terminates HTTPS connections so as to retrieve plain-text HTTP sessions from encrypted flows.

Separating virtual execution environments into PEXes and GEXes manifests MAPPER's balance between flexibility and performance of the system. Providing each user a dedicated PEX enables adopting user specific net-apps, enables security and privacy stemming from traffic separation and provides support for a user to program and customize her own net-apps. On the other hand, the GEX enables MAPPER to avoid the overhead of running several instances of the same net-app in multiple PEXes.

3.2 Virtualization Layer

The virtualization layer that supports the creation of PEXes and GEXes in MAPPER is built on top of the FROG (Flexible and pROGrammable) edge router platform [11, 2]. The high level architecture of FROG is depicted in Figure 1. A PEX is a FROG execution environment that receives/sends data from the underlying network hypervisor through an high speed communication channel based on shared memory, while GEXes are reached through standard TCP/IP sockets. A GEX is an isolated container that shares the TCP/IP stack with other GEXes and the rest of the host (including the hypervisor) and uses standard TCP/IP primitives to communicate with the other components. The hypervisor forwards incoming packets to proper PEXes for user specific processing. In the FROG implementation deployed in this work the source and destination MAC addresses of a packet are used by the hypervisor to determine which PEX the packet should be forwarded to. Packets are processed by net-apps running in PEXes and then they, or their content as extracted by the net-apps, are processed by GEXes, if there are any. After that, the hypervisor relays the processed packets out of MAPPER.

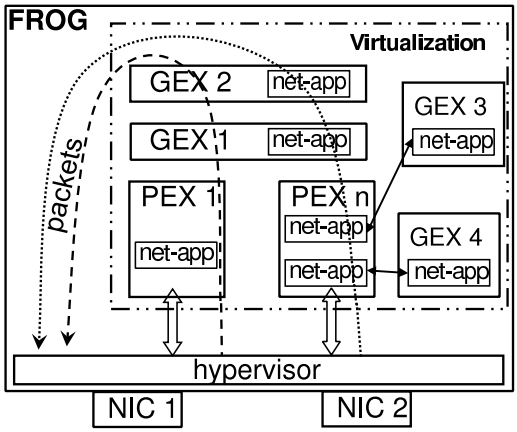

Figure 1: Virtualization layer of MAPPER.

3.3 Network Applications (Net-apps)

We have developed a broad range of net-apps for MAPPER. Some of them are quite generic and can be adopted in a variety of scenarios. Examples of such generic net-apps include a *network monitor* and a *DNS filter*, which collects traffic statistics and filters out malicious DNS queries, respectively. In this paper we focus on net-apps related to the enforcement of policies on mobile apps.

Man-in-the-middle Proxy (MiMP) Net-app: To provide the visibility into encrypted traffic from mobile apps, MAPPER uses a GEX to run a TLS-capable proxy service. For this purpose we used `mitmproxy` [3], which is a powerful and customizable transparent HTTPS proxy. `mitmproxy` relies on a TCP/IP stack to provide TCP end-point functionalities, which are in turn required to support the TLS sessions terminated by the proxy. Rather than including a TCP/IP protocol stack implementation in the GEX, the proposed design leverages an additional virtual network interface `tap0` bound to the OS TCP/IP stack. The `mitmproxy` running in a GEX uses its services through the common socket interface provided by the OS. Leveraging its full visibility into user's traffic, the MiMP net-app also extracts relevant information from traffic required by other modules of the system, such as the Mobile App Filter net-app presented later.

MiMP Bridge Net-app: A user's PEX runs a MiMP Bridge net-app to interact with the MiMP net-app running in a GEX. Once loaded into a user's PEX, the MiMP Bridge net-app forwards all the (encrypted) user's traffic toward the MiMP net-app, which returns a selected portion of the plain-text form of the encrypted traffic to another net-app. In the enforcement of per-user policies based on mobile apps, the returned plain-text form data is consumed by the Mobile App Filter net-app to enforce mobile app usage policies. It is worth highlighting that encrypted traffic is analyzed only for users that have the MiMP Bridge in their policy.

Mobile App Identification (MAI) Net-app: This MAPPER net-app runs in a GEX and identifies the mobile app generating network flows to enable the definition and enforcement of policies at mobile app granularity. The MAI net-app comes preloaded with a rule set [4, 14] that extracts distinguishing mobile application features from individual flows. These per-flow features are then matched with a rich signature table to identify the app responsible for the flow. In addition, the MAI net-app also assigns device (e.g. iPhone, iPod, Samsung Galaxy) and platform (e.g. iOS and Android) labels to classified flows. Last but not least, MAI uses category trees to map each mobile app into a set of carefully pre-selected categories, such as social, gaming, etc. Therefore, network monitoring and management policies can be formed taking into account not only individual apps, but also broad app categories, platforms and device types; or any combination thereof.

Mobile App Filter Net-app: The service provided by the MAI net-app is consumed by the *Mobile App Filter* net-app running in a user's PEX to enable the enforcement of her specific policies on mobile app usage. For each HTTP/HTTPS bi-directional flow, the Mobile App Filter receives an extract of the flow from the MiMP and contains information needed by MAI to identify the appID of the originating mobile app. The XML formatted extract is passed from MiMP to Mobile App Filter and then to MAI through REST APIs, also used to return the appID from MAI to Mobile App Filter. The policy enforced for a user can be blacklisting a set of mobile apps, or even the entire categories of mobile apps (e.g. social, gaming, etc). Once Mobile App Filter determines that the originating app as identified by an appID is in the blacklist, the response to the REST API previously called to the Mobile App Filter indicates to the MiMP the corresponding flows should be blocked.

3.4 Policy Enforcement Example

We present an example of policy enforcement to show the net-apps involved and the flow of information among them. Every time a new user is connected to a MAPPER wireless AP, the user is required to authenticate via a captive portal web UI. Upon a successful authentication, a dedicated PEX is instantiated by MAPPER and the predefined policy, in the form of net-apps, is loaded into the PEX. Figure 2 shows an example of the net-apps deployed and the flow of information among them, where MAPPER enforces a policy to require visibility into both HTTP and HTTPS traffic, and to blacklist a set of mobile apps.

Each packet from the user device is classified by the hypervisor and forwarded to that user's PEX to be processed by a chain of net-app. First, the MiMP bridge net-app hijacks all HTTP and HTTPS traffic towards the MiMP GEX. The hijacking is done by changing the destination IP address to the one bound to virtual interface `tap0`, on which the MiMP is listening for incoming connections. The packets with modified destination IP address are then sent back to the hypervisor that forwards them to the `tap0` interface of the MAPPER machine, where they are received by the MiMP. For each packet flow, the MiMP bridge net-app also informs the MiMP about the original 5-tuple (i.e. $\{sIP, sPort, dIP, dPort, proto\}$).

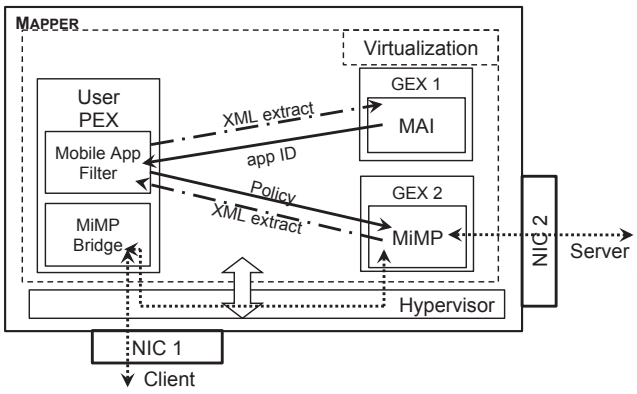

Figure 2: Information flow among net-apps.

The 5-tuple information is consumed by the MiMP to build a map M_{dIP}, which maps $\{sIP, sPort, dPort, proto\}$ to dIP. MiMP uses map M_{dIP} to recover the original destination IP of an incoming connection, which is needed to create an outgoing TCP connection to the proper server and forward to it the HTTP/HTTPS session.

If an incoming connection is HTTP, the MiMP acts as a conventional HTTP proxy, which opens a new connection with the original web server and fetches the content. For a HTTPS incoming connection, the MiMP terminates the secure TLS tunnel and joins it with a new one to the external web server. The MiMP is able to terminate the secure tunnel because during the TLS negotiation phase it returns to the client application a self signed certificate of the web server, rather than the original one. The client host must have the MiMP certificate installed as a trusted Certification Authority, which might be performed in the initial authentication phase, or the user is required to accept the MiMP signed server certificate.

As shown on the right hand side of Figure 2, the HTTP request to the original web server is sent through the OS TCP/IP stack without hitting the hypervisor of the MAPPER. The first HTTP message sent by mobile apps is allowed to reach the web, even if the mobile app should be blocked. We deliberately allow this because the server response could contain information useful for identifying the app. We believe that in general this does not represent a security threat since the first HTTP request/response is usually designed for the app to probe the server and render the app user interface.

Once the MiMP receives the response from the web server, it extracts relevant data from the request and the response, encodes it in XML form, and sends it to the Mobile App Filter net-app, which then queries the MAI using the XML extract. The MAI returns information related to the mobile app that generated the traffic, including the unique app ID, category of the app, and OS of the app. The Mobile App Filter matches this data against the blacklist in the user policy to check if the app is permissible or not. The match result is sent back to the MiMP, which will act according to the result, by sending back to the client either the response or an HTTP error message.

The HTTP response is held by the MiMP while it waits for feedback from the Mobile App Filter. Performance can be improved and latency reduced at the expenses of a small reduction in app identification probability by having the MiMP producing the XML extract from HTTP requests only. Hence, while the proxy contacts the server and awaits for a response, the MAI net-app can work with the XML extract distilled from the HTTP request to provide the Mobile App Filter with the information on the mobile app. The Mobile App Filter matches it against the policy and communicates the outcome to the MiMP so that the latter is possibly ready to forward the HTTP response as soon as it is received.

Note that the visibility into HTTPS traffic is optional and can be disabled by user policy. Without the requirement to inspect HTTPS traffic, the MiMP bridge net-app forwards to the MiMP net-app only HTTP traffic, while HTTPS one is left unchanged.

4. EVALUATION

4.1 Implementation and Experiment Setup

We have implemented a prototype of MAPPER using a small form factor machine – an Intel *Next Unit of Computing* box, with an Intel Core i3-3217U CPU, 8G RAM, running Ubuntu Linux. The per-user PEXes are FROG virtual machines realized as standard Java Virtual Machines (JVM). The MAI runs inside a FROG VM as a GEX net-app. At present, the MAI net-app is capable of identifying more than $250K$ mobile apps spanning the iOS and Android platforms. The MiMP GEX runs an open source python implementation available at [3]. To ease the prototyping, we did not port MiMP into Java. Instead, we did some small changes to the python code to enable the interaction between MiMP and other net-apps.

We setup a small testbed as our experiment environment to evaluate the performance of MAPPER. In the testbed a MAPPER system works as a wireless AP serving smartphone and laptop devices. To easily automate some experiments, the laptop runs `curl`, a customizable command line web client, to emulate various mobile apps.

4.2 Single-user Tests

We start with testing the performance of MAPPER in serving only one user. To provide the baseline for fair comparison, we test three different settings of the access point machine. We configure the machine as (1) a standalone software AP running `hostapd` [1], (2) a standard FROG box running one per-user VM, and (3) a MAPPER system running the MAI GEX, the MiMP module, and one per-user PEX. We refer to those three configurations as AP, FROG, and MAPPER, respectively. For the MAPPER setting, the user policy is set to allow all traffic from user's mobile apps.

To isolate our experiments from other interferences, we connect a web server and the access point machine to the same wired LAN. Then we run the command line web client to emulate the activity of a mobile app X by sending 500 HTTP requests to fetch 500 different files from the web server. While the tests are running, we measure a number of metrics at both the laptop client and the access point machine.

We first evaluate the network throughput metric of the system by downloading 500 files from the web server, where each one is of size $1MB$. Figure 3 shows the average throughput measurements at the laptop's wireless interface. We can see that the current MAPPER implementation indeed has lower network throughput as compared to AP and FROG settings, i.e., the throughput is 20% lower. The main reason is that our current implementation requires the Mobile App Filter net-app and the MiMP GEX to interact synchronously, which results in considerable blocking and waiting among them. We are working extensively on removing this bottleneck from the system.

We also record the CPU and memory usage at the access point machine, and report the results in Table 1. The results show that RAM is a more critical resource than CPU in our current prototype implementation, since MAPPER consumes much more RAM than the other two settings. A closer look into each module of MAPPER shows that the MAI GEX alone consumes almost $1.8G$ of memory,

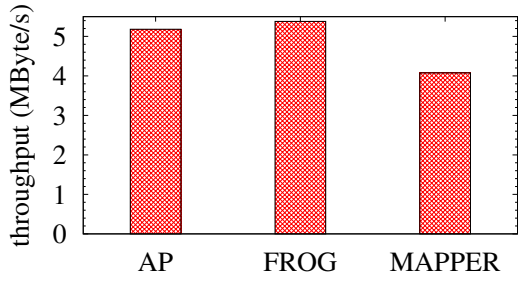

Figure 3: Network throughput test results.

because the app identification engine uses several big hash table data structures.

	CPU peak	CPU ave	RAM peak	RAM ave
AP	3.4%	1.82%	490 MB	487 MB
FROG	61.3%	17.88%	1241 MB	1190 MB
MAPPER	62.6%	36.5%	4080 MB	3698 MB

Table 1: CPU and RAM usage comparison.

We next evaluate how much additional delay added by MAPPER to user's traffic. We measure the time interval between the web client sending out a request and receiving the reply from the web server, and refer to that interval as the *response time*. For each setting of the access point machine, we measure the response time of 500 requests, where each one fetches a $1KB$ file from the web server, and plot their cumulative distribution in Figure 4. As expected, users do experience longer response time due to the additional processing of user traffic by our current implementation of MAPPER. The response time is less than $500ms$ in more than 80% of the cases. The average additional delay is around $300ms$.

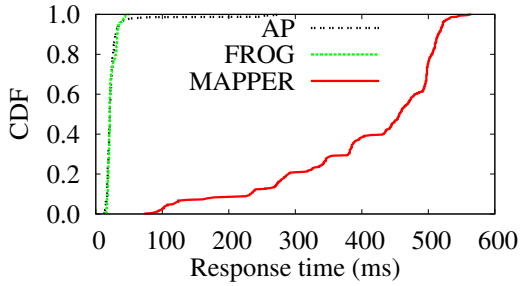

Figure 4: CDF of the response time for three settings of the access point machine.

To evaluate how the additional overhead introduced by MAPPER affects user experience, we have recruited several volunteers to connect their smartphones with MAPPER and use their mobile apps for a few minutes. According to the feedback, most users do not have any noticeable user experience degradation in surfing the web, checking online social network updates, etc.

4.3 Multi-user Tests

After having a good understanding of MAPPER's performance in single user scenario, we proceed to evaluate the scenario where multiple users connect to the MAPPER device. To that end, we simulate the presence of multiple clients by allocating various number of user PEXes in MAPPER and measuring the memory usage of the MAPPER machine. We focus on memory usage because it is the dominating factor that constrains MAPPER from scaling to large number of users. The results are reported in Figure 5.

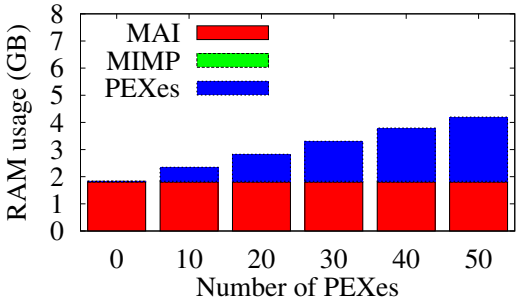

Figure 5: RAM consumption with growing number of users.

The benefit of having GEXes in MAPPER is obvious as shown in Figure 5. Because we don't replicate MAI and MiMP into each user's PEX (although MiMP consumes very little memory, around $30MB$), the memory footprint of each user PEX can be limited to around $50MB$ when the PEX is not processing any packets. Even after scaling the number of PEXes to 50, the MAPPER machine uses only half of its memory ($4G$ out of $8G$). We believe that most WiFi network deployments do not plan to have one AP serving more than 50 users.

In order to test how the MAPPER behaves in a more realistic scenario, we also emulate mobile devices using a public Internet web service on encrypted HTTPS channel. We connect two laptops to MAPPER to emulate that two different users (each one has a dedicated PEX) are using mobile app X to search at `https://www.google.com`. To maximally stress the system, both users' policies are configured to be allowing app X and hijacking that app's encrypted traffic. We simultaneously set both laptops to emulate app X sending out 500 search queries. When the test is running, we measure the CPU and memory usage at the MAPPER machine, as well as the network throughput and response time at both laptops. To provide a fair comparison baseline, we also repeat the same experiment using only one laptop to emulate one single user.

	CPU	RAM	throughput	response time
1 user	16.13%	3261 MB	104 Kb/s	778.8 ms
2 users	21.57%	3392 MB	102 Kb/s	751.6 ms

Table 2: Averaged performance measurements when users are using mobile app X to perform a web search at `https://www.google.com`. Throughput in the 2-user test is the one measured at one client device.

Table 2 presents the test results. We can see that when serving two users instead of one, MAPPER consumes slightly more CPU and RAM resources, without noticeable degradation in throughout and response time. Because the web server is on Internet instead of connected to the same LAN, the response time in Table 2 is larger than that shown in Figure 4. The larger response time and lower network throughput result in slower traffic rate. Therefore, the CPU and RAM usage are lower than those presented in section 4.2.

5. CONCLUSION AND FUTURE WORK

This paper presents the MAPPER system that seamlessly integrates multiple techniques to facilitate enforcing policies on increasingly flourishing mobile apps running on portable devices. We described the design of MAPPER system in detail and evaluated its performance using a small scale testbed network. We showed that network function virtualization provides MAPPER the flexibility to

implement fine-grained policies with acceptable performance overhead. MAPPER is a work-in-progress system and our future studies will be directed towards performance improvement on various aspects of the system. First step in this is to improve the performance of MiMP. Currently the MiMP GEX represents a severe bottleneck. We strongly consider that a Java multithreading implementation of MiMP can scale up the overall performance. Besides, we plan to use shared buffers for faster inter-module communications and deploy a web-cache as a net-app to improve performance.

6. ADDITIONAL AUTHORS

Additional authors: Ruben Torres (Narus, Inc.) and Antonio Nucci (Narus, Inc.).

7. REFERENCES

[1] hostapd and wpa_supplicant. http://hostap.epitest.fi/.

[2] I. Cerrato, M. Pramotton, and F. Risso. Moving applications from the host to the network: Experiences, challenges, and findings. In *IEEE Workshop on Mobile Cloud Networking (MCN)*, 2013.

[3] A. Cortesi. mitmproxy - a man-in-the-middle proxy. http://mitmproxy.org/.

[4] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song. NetworkProfiler: Towards Automatic Fingerprinting of Android Apps. In *Proceedings of IEEE INFOCOM*, 2013.

[5] N. Egi and et al. A platform for high performance and flexible virtual routers on commodity hardware. *ACM SIGCOMM Computer Communication Review*, 2010.

[6] A. D. Keromytis and J. L. Wright. Transparent network security policy enforcement. In *Proceedings of USENIX ATC*, 2000.

[7] K. W. Miller, J. Voas, and G. F. Hurlburt. BYOD: security and privacy considerations. *It Professional*, 14(5):0053–55, 2012.

[8] O. Morandi, F. Risso, P. Rolando, S. Valenti, and P. Veglia. Creating portable and efficient packet processing applications. *Design Automation for Embedded Systems*, 15(1):51–85, 2011.

[9] B. Morrow. BYOD security challenges: control and protect your most sensitive data. *Network Security*, 2012(12):5–8, 2012.

[10] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. Corrêa, S. C. de Lucena, and M. F. Magalhães. Virtual routers as a service: the routeflow approach leveraging software-defined networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, pages 34–37. ACM, 2011.

[11] F. Risso and I. Cerrato. Customizing data-plane processing in edge routers. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 114–120. IEEE, 2012.

[12] R. Sailer, T. Jaeger, X. Zhang, and L. Van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004.

[13] Symantec. Internet security threat report 2014. http://www.symantec.com/security_response/publications/threatreport.jsp.

[14] A. Tongaonkar, S. Dai, A. Nucci, and D. Song. Understanding mobile app usage patterns using in-app advertisements. In *Passive and Active Measurement*. Springer, 2013.

[15] Q. Xu, T. Andrews, Y. Liao, S. Miskovic, Z. M. Mao, M. Baldi, and A. Nucci. FLOWR: A Self-Learning System for Classifying Mobile Application Traffic. In *Proceedings of ACM SIGMETRICS*, 2014.