INVITED PAPER *Special Section on Network Systems for Virtualized Environment*

Application Specific Slicing for MVNO through Software-Defined Data Plane Enhancing SDN

Akihiro NAKAO[†a)], **Ping DU**[†b)], ***Members***, ***and*** **Takamitsu IWAI**[†c)], ***Nonmember***

SUMMARY In this paper, we apply the concept of software-defined data plane to defining new services for Mobile Virtual Network Operators (MVNOs). Although there are a large number of MVNOs proliferating all over the world and most of them provide low bandwidth at low price, we propose a new business model for MVNOs and empower them with capability of tailoring fine-grained subscription plans that can meet users' demands. For example, abundant bandwidth can be allocated for some specific applications, while the rest of the applications are limited to low bandwidth. For this purpose, we have recently proposed the concept of *application and/or device specific slicing* that classifies application and/or device specific traffic into slices and applies fine-grained quality of services (QoS), introducing various applications of our proposed system [9]. This paper reports the prototype implementation of such proposal in the real MVNO connecting customized smartphones so that we can identify applications from the given traffic with 100% accuracy. In addition, we propose a new method of identifying applications from the traffic of unmodified smartphones by machine learning using the training data collected from the customized smartphones. We show that a simple machine learning technique such as random forest achives about 80% of accuracy in applicaton identification.

key words: *Software-Defined Networking (SDN), Network Functions Virtualisation (NFV), network virtualization*

1. Introduction

Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) have recently caught attentions from industries as technologies for reducing capital expense (CAPEX) and operational expense (OPEX), where software-defined programmable network equipment dispenses with high maintenance cost of hardware appliances and enables rapid revisions of functionalities and the automation of operation and management (OAM) of network. While SDN primarily focuses on the programmability on the control of networking, NFV aims at implementing data processing functions in software on top of virtual machines (VMs) especially that exist today as hardware network appliances. Data packets can be *programmatically redirected* by SDN and can be *programmatically processed* by NFV.

We have recently posited that *software-defined data plane*, i.e., arbitrarily defining data plane by software programming, significantly enhance the synergy between SDN and NFV [8]. In carefully designed sandboxes such as virtual machines inside network equipment, we should be able to enhance the data plane functionalities, e.g., those related to OAM, and publish the SBI (Southbound Interface) for controllers to use them. Such enhancement is only recently discussed in a few research projects [1], [4]. Also, NFV is so far limited to implementing network appliances in software, and deals neither with crafting new protocols nor with OAM functionalities. Since the current SDN's data plane is not so much flexibly programmable because it is still often implemented in hardware, enhancing SDN with software-defined data plane would fill the gap in the current NFV.

We have also proposed the application of software-defined data plane to defining new services for Mobile Virtual Network Operators (MVNOs) that obtain network services from mobile network operators and resell network services to customers at their own prices without owning the wireless network infrastructure on their own [9]. More specifically, we have defined *application and/or device specific slicing* that classifies application and/or device specific traffic into slices and applies fine-grained quality of services (QoS) for MVNO, introducing various applications of our proposed system. There are a large number of MVNOs proliferating all over the world and most of them provide low bandwidth at low price. We have proposed a new business model for MVNOs to empower them with capability of tailoring fine-grained subscription plans that can meet users' demands, for example, abundant bandwidth is allocated for some specific applications, but the rest of the applications are limited to low bandwidth.

In this paper, we show our prototype implementation of our proposal in [9] in the real MVNO connecting customized smartphones so that we can identify applications from the given traffic with 100% accuracy using deeply programmable nodes developed in FLARE project [1]. In addition, we propose a new method of identifying applications from the traffic of unmodified smartphones by machine learning using the training data collected from the customized smartphones. We show that a simple machine learning technique such as random forest [5] achieves about 80% of accuracy in application identification.

The rest of the paper is organized as follows. Section 2 introduces our design decisions for enabling application and/or device specific slicing utilizing programmable software-defined data plane. Section 3 discusses various applications of our proposed system. Section 4 introduces our prototype implementation and experiments using our programmable network node architecture called FLARE and shows our preliminary evaluation on application identifica-

Manuscript received August 11, 2015.

[†]The authors are with The University of Tokyo, Tokyo, 113-0033 Japan.

a) E-mail: nakao@iii.u-tokyo.ac.jp

b) E-mail: duping@iii.u-tokyo.ac.jp

c) E-mail: iwai@nakao-lab.org

DOI: 10.1587/transcom.E98.B.2111

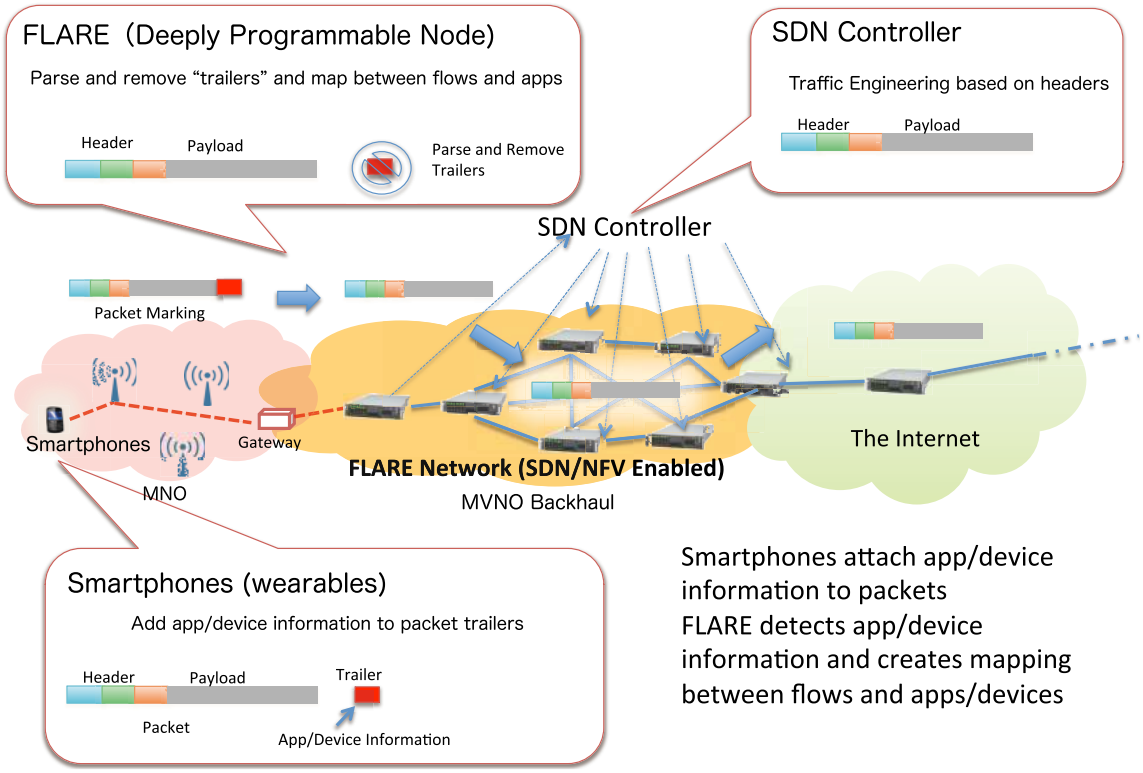


Fig. 1 Application/device specific slicing.

tion using machine learning. Finally, Sect. 5 discusses future work and Sect. 6 briefly concludes.

2. Design

This section introduces our preliminary design for application and device specific slicing to enable Software-Defined Networking and Network-Functions Virtualization for MVNO.

2.1 Overview

In order to realize application and/or device specific slicing, as shown in Fig. 1, we have designed *trailer slicing* [1] where *meta information* on applications and devices at the end of packets (as discussed in more detail in Sect. 2.3.1.) Note that in our design, the meta-information may include many other kinds of information, but for the sake of brevity, we limit its scope to applications and devices within this paper.

In generic trailer slicing, each packet may carry trailer bits containing meta information of the packet, such as from which application process and/or from which device a packet has been transmitted. For example, we install our software on smartphones for capturing the very first packet an application transmits, i.e., a TCP SYN packet when the application establishes a TCP session, and then for attaching trailer. In more detail, we capture the header information of a TCP SYN packet and examine the process table and the socket table of the operating system to look for a corresponding application process that uses the flow space (such as IP addresses and port numbers) and attach the information regarding the application process such as a name and status as a trailer. Note that this approach can be easily applied also any other transport protocol such as UDP. Also we can attach device information as well as that of application.

A programmable node, e.g., FLARE (explained in 4.1) at the gateway to the MVNO backhaul network detects the trailer attached to the unusual TCP SYN (with non-zero payload size) or the packet that uses a flow space for the first time, and decodes/removes the information in the trailer. It also observes the flow space information of the packet at the same time and maps the information on the application processes and that of the flow. When the SDN controller can receive this mapping information, the subsequent packets can be controlled by the SDN switches along the route to the destination according to the flow space information associated with application/device information. For example, we can perform QoS traffic control such as bandwidth throttling for particular applications/devices using the traditional flow-based traffic control.

2.2 Filling a Gap between Application/Device and Network Programming

Although SDN and NFV are considered useful tools for programmable networking, we observe a gap between develop-

ment of applications, services and devices, and that of programmable networking, mainly caused by the gap between abstractions defined in two worlds.

In the current Internet, applications and services implemented on end systems use socket interface to utilize services provided by the communication infrastructures. Since socket interface provides clear separation between end-systems and networking, the context of applications, services and devices are dismissed when packets are transmitted into the network. In other words, unless performing deep packet inspection (DPI) on packets or inferring from various characteristics such as packet length and timing, it is difficult to tell which application context sends/receives those packets.

Operating systems on top of end-systems use processes and threads as abstraction for programming applications and services. The current SDN networking equipment uses flow information as abstraction for programming network. In a sense, our proposal for application/device specific slicing bridges the abstractions used in operating systems and programmable networking.

2.3 Slicing Mechanism

2.3.1 Trailer-Slicing

The idea of trailer slicing is to attach a slice identifier at the end of the packets under the agreement of the existence of such bits among the users of the infrastructure, for example in mobile backhaul networks, at cloud data centers, and in any other administrative domains where the agreement may be established. As briefly explained in 2.1, a slice identifier may not just be an explicit number, but can be the meta information to identify a slice such as application name or device type under the agreement.

In SDN, we have been using the header information to define a slice, specifically, so-called flow information, which is a combination of MAC addresses, IP addresses and port numbers. However, when we consider cooperation between operating system entities and networking, we conclude that we should use a more straightforward identifier, such as a process name, an application name and a device type, etc. We can define a name space so that within the name space a slice can be identified uniquely, e.g., `com.android.google.youtube` in case of the name space for process names in the Android operating system.

The idea of trailer slicing is similar to MPLS in that we use bits (that can be view as label) for switching, but the difference is that the position of bits in layers and in packets and the length of the bits. We intentionally put a slice identifier at the end of packets. With adjustment of header fields in L3 and/or L4, we can get packets through with trailers through the existing network equipment, since they treat trailers as L7 data bits. Of course, we need to remove trailers before packets reach the destination, but that should be taken care of by the agreement of trailer slicing among administrative domain.

One may argue that we could use header option fields instead of a trailer for storing a slice identifier. However, there is a risk that non-standard header options may be removed or may cause network equipment to malfunction. Also option fields may be in short of bits, flexibility and extensibility. To avoid pressing header handling on the part of legacy network equipment, we decide to use a trailer since all the network equipment along the route of a packet treats a trailer as a part of payload data, so it keeps preserved till it gets parsed and removed. However, our scheme could be easily implemented in header options of course, when the concerns above are not an issue.

Note that not all packets need to carry trailers, although such design is certainly possible. As long as we agree on which packet in a flow carries a slice identifier, we can establish mapping between the traditional flow information and the slice identifier in network equipment. After the mapping is created, from then on, flow information could be used for the slice identifier.

As an aside, there is an interesting use case of trailer slicing called TagFlow [6], where we push expensive complex classification to the edge of the network and use one field trailer to simplify the classification at the core of network. In TagFlow, every packet is expected to carry a trailer.

2.3.2 TCP-SYN Piggy-Backing

Some may argue that piggy-backing data in TCP SYN may render incompatibility and security issues. However, such unusual piggy-backing is not uncommon today. For example, Google does this in TCP Fast Open (TFO) [10] for the different purpose than ours, where they attempt to reduce the number of packets and the delay in three-way handshaking, storing “cookies” in newly emitted TCP SYN’s payloads for already authenticated end systems via the past three-way handshakes.

2.4 Signaling between End-Systems and Network

2.4.1 Out-of-Band Signaling

Even if applications keep track of their flow information, they need to let the SDN controller know the flow information out of band, that is, besides the application data traffic, they must open control channels to convey such flow information to the SDN controller so that they may be able to control their flows. This approach is prohibitive for a large number of small devices such as smartphones and sensors since it may become significant overhead for them.

2.4.2 In-Band Signaling

We propose a method to modify operating systems of the end systems such as smartphones, so that we can find application process information and convey such information through an in-band communication. Our prototype system attaches the application process information as a trailer on

the part of the end systems, and decodes the information in it then removes it on the part of the programmable node located in the backhaul, ideally at the first hop from the gateway from a mobile network operator (MNO). In this way, we learn the mapping of the information on application processes and flows and inform the SDN controller so that subsequent nodes can just perform the conventional flow-based traffic control.

2.5 Wide-Area End-System Management

There are two main challenges in designing a wide-area end-system management: security and reliability. The management system should have identification and authentication mechanisms so that it cannot be easily hijacked. Reliability means that the control messages should be able to reach to end-system independent of the network/device status.

We address the above two challenges with cloud services such as Google Cloud Messaging (GCM) service. Although from now on, we explain our design for wide-area end-system management using the GCM service, for the sake of ease of understanding of the requirements in our design, any other similar cloud service system can be used in our system design.

When we register with GCM service, we are issued with a *Sender ID* as well as a *Sender Auth Token*. The former enables smartphones to talk to the GCM server while latter enables our management system to talk to the GCM server. Since a smartphone may be in sleep or offline mode that cannot receive/process a control message from our management system instantly, the management system doesn't send control messages to smartphones directly. Instead, it sends a message via the GCM server. The GCM server can enqueue and store the message in case a smartphone is offline. When it is online, the GCM server checks the message queue and sends stored messages to the smartphones. The GCM server can be also configured to wake up smartphones in sleep on receiving GCM messages.

Figure 2 illustrates the basic work flow of a our designed wide-area end-system management. It consists of two parts: registration (red arrows) and messaging (blue arrows).

Steps 1-2: When a smartphone boots up, it sends a registration request to the GCM server. After the successful registration, the GCM server issues a device unique *registration_id* back to the smartphone.

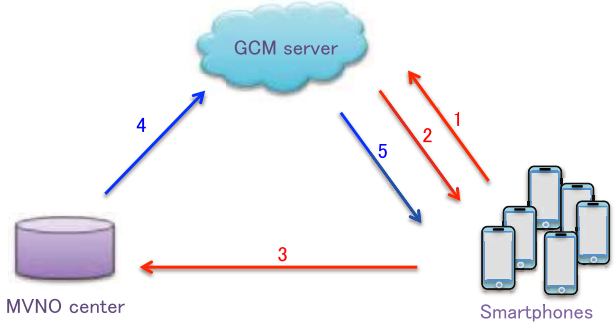


Fig. 2 Wide-area end-system management architecture.

Step 3: After receiving the *registration_id*, the smartphone sends its *registration_id* with *IMEI* to end-system management system (MVNO center). The MVNO center stores the information in its database. We use IMEI to identify a smartphone user because we find that some carriers may reassign IP address to a smartphone when the phone is in sleep. A phone number is not a good candidate for the identifier since a user may unplug or change his/her SIM card.

Steps 4-5: When MVNO center needs to send a GCM message to a smartphone, it first looks up the database and gets the *registration_id*. Then, it sends a message to the GCM server together with *registration_id*. The GCM server forwards the message to a corresponding smartphone. To check the smartphones' states, the MVNO center sends keepalive GCM messages to the smartphones periodically.

2.6 Deep Data Plane Programmability

We should note that in order to enable application/device specific slicing for MVNO, data-plane functionality must be extended from the current SDN model where data-plane elements have limited pattern match capabilities and too few actions. Especially note that the manipulation of the packet trailer at Layer 7 (L7) is largely missing from the current SDN data-plane elements and the extension to support such manipulation is useful to enable new applications.

3. Applications

3.1 Traffic Engineering

Traffic engineering such as Quality of Service (QoS) and route/switch control for specific applications and devices is the immediate application of our proposal in this paper. We can create slices according to (1) *application names*, (2) *application processes*, (3) *device types*, and (4) *device status/location*, etc. However, it is obviously possible to extend a trailer to include much more information about applications, devices, the context of usage of them, etc.

3.2 Value-Add Services

After classifying application and/or device specific traffic into slices, we can apply NFV virtual functions to perform useful data processing such as *compression and decompression* and *packet caching*, etc. This application helps differentiating competing applications such as web browsers. For example, a certain browser can benefit from installing transparent data cache near smartphones, while other browsers may not. We expect more and more applications on smartphones can be empowered by small, yet smart functionalities embedded in NFV for aiding the operations of the applications.

3.3 In-Network Security

Another interesting example application is in-network security. *Malware containment in a slice* is one example application. In our prototype, as long as malwares on the smartphones transmit packets, we can catch the traffic from those processes and contain the traffic into a slice, by examining the application process names associated with flows. Our prototype system even raises alerts to smartphones that they may have accidentally installed malwares on them once their traffic get detected.

Also, *in-network parental control* is another example application in the security area. Usually, parents would like to restrict the usage of applications on their childrens' smartphones by installing parental control software on them. However, in most of the cases, those applications may be removed easily by the children. In our system, since application and device specific traffic can be classified into a slice, we can easily set policy and control bandwidth such traffic. For example, the traffic from a specific application on a specific device can be controlled on the part of network, not on the device, for a determined period of time. The parental control enabled by this mechanism is not easily removed by the hands of the children.

3.4 Big-Data Analysis

Neither capturing nor deeply inspecting users' traffic are allowed in several countries such as Japan. However, MVNO operators are interested in collecting application specific bandwidth usage to provide more fine-grained subscription plan. We intentionally design our system so that the privacy of user data (L7 payload data) may not be infringed. If users are fine with their application usage being collected, we believe we can alleviate the dilemma between MVNOs' demands for bandwidth usage data and users' privacy. Most of the related work for identifying applications from the traffic trace relies on deep packet inspection (DPI) of the user data, which may not work if DPI is restricted by law or the packet payload data is encrypted.

There are lots of MVNOs proliferating in Japan, but most of them offer low bandwidth at cheap price, which causes fighting for selling ever-lower-cost subscription plans among those MVNOs. We believe an MVNO may be able to create a fine-grained and tailored subscription plan that can meet users' demands, for example, provisioning bandwidth for some specific applications, but the rest of the applications are limited to low bandwidth. In order to come up with viable subscription plans, application traffic analysis becomes a key.

3.5 Application-Specific QoS For Unmodified Smartphones

Another possible application of our proposed system is application specific QoS for unmodified smartphones. We propose a method for inferring applications from given traffic in real-time by machine learning using reliable training data generated by the customized smartphones as discussed so far.

In the existing research, the proposed method often captures traffic data for a long time, e.g., for several months and then characterizes it using deep packet inspection (DPI), thus has limitations such as costly generation of training data (e.g., large volume of traffic capture data), privacy violation caused by inspecting packet payloads via DPI, and difficulty of DPI due to encryption. As a result, they end up with unreliable training data for classifying traffic.

In contrast, our method relies only on a small number of customized smartphones that generate the mapping between characterization of a given flow and the application that transmit the flow. We characterize a given flow using only stateful behaviors such as means and variations of the packet length or in header information in a train of packets contained in the flow, without even looking at the payload of packets. Therefore, our method compensates for the drawbacks of the existing methods in that (1) we can generate the training data in realtime (less cost), and (2) we neither violate the privacy nor have difficulty of DPI in encrypted flows.

Figure 3 illustrates our proposed method. We capture the training data, that is, the packets tagged with application information, transmitted from a small number of customized smartphones, store them for a certain period and characterize them in a batch using the stateful information of the flow explained above. Thus we construct a classifier according to application kinds using the characterization of the training data. Then, we classify the traffic that is not tagged with the application information.

According to our test run of the proposed method, our proposal can analyze the real traffic captured at an MVNO and successfully identify mobile applications with about 80% credibility when the learning period is 5 days using a simple machine learning technique called random forest [5], an ensemble learning method where a multitude of decision trees are constructed at training time, each of which is obtained by a randomly sampled subset of training data.

Although our method seems promising, we have lots

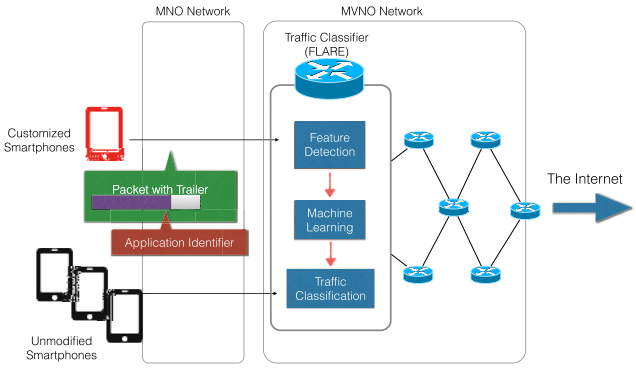


Fig. 3 In-network application classification through machine learning.

of challenges for the near future, such as increasing the accuracy that is sufficient for application-specific QoS and investigating which machine learning algorithm is best for application identification, such as deep learning and support vector machine.

In Sect. 4, we introduce more detailed experiment results regarding this application.

4. Implementation and Evaluation

4.1 FLARE

FLARE is a deeply programmable network node architecture [1] utilizing a hybrid of computational resources, such as network processors, general purpose processors, (and optionally GPGPU) hierarchically to extend data plane processing functions easily by software program.

FLARE tackles three research challenges, (1) ease of programming, (2) reasonable and predictable performance, and (3) enabling multiple concurrent isolated logics. For (1), we introduce Toy-Block networking programming model [7] to facilitate drag and drop data plane programming. For (2), we combine of high-frequency small-number-core processors for control and management functionalities, and low-frequency many-core processors for massively parallel processing for a large number of flows. And finally, for (3), we employ a lightweight resource virtualization technique called resource container for isolation of multiple logics. For the best isolation, we decide to partition many cores into groups and deploy a resource container per group.

The goal of FLARE is similar in spirit to that of OpenDataPlane [3], especially in that the purpose is to flexibly and easily extend data plane. However, the key difference is that we consider isolation of resources to support multiple concurrent data plane logics via virtualization. For example, FLARE program multiple concurrent logics such as OpenFlow 1.0 and OpenFlow 1.3 data plane elements in isolated execution environments.

4.2 Prototype Implementation

Utilizing FLARE prototypes, we have implemented our prototype system to enable application and/or device specific slicing for MVNO as shown in the overview of our design depicted in Sect. 2.1. We have developed Android smartphone software to enable trailer slicing, i.e., embedding a slice identifier at the trailer of TCP SYN packets and QoS traffic engineering per slice on our FLARE platform [1]. We have discovered that we can use TCP SYN trailers unless ISPs do not filter unusual TCP SYN in fear of SYN Flooding, which is not really performed in most MVNO services of today.

In implementing our prototype, we reconsider southbound API (SBI) for your application. As reviewed in Sect. 2.2, we believe application users and developers, process-based traffic control is more natural than flow-based one. Extending the Openflow model, the right abstraction for programming in this case may be one such as,

<Application/Device><Action><Stat>,

instead of

<Flow><Action><Stat>,

although we may not have to follow OpenFlow's convention for programming abstraction and also one could rather define one's own programming abstraction, as long as it is open and published as an API.

We also jointly operate our prototype system with an ISP in Japan with 65 Android phones and successfully demonstrate our prototype system works on top of an MVNO. We plan to extend our experiments to enable various application ideas shown in Sect. 3. Note that the same prototype but with WiFi network has been demonstrated successfully at various venues such as GEC20 [2]. We believe that empowering MVNOs with application/device specific traffic engineering would become the norm of the next generation MVNO business.

4.3 Characteristics of MVNO Network

In order to manage MVNO effectively, it is desirable to understand the traffic pattern and the user behaviors. In this section, we show the statistics collected from the real MVNO with FLARE programmable nodes deployed by connecting customized smartphones.

Usually, MVNO may not be able to obtain the accurate classification of the total amount according to the transmitting applications. However, our proposed system can show the breakdown of application traffic with 100% accuracy as follows.

Figure 4 shows the fractions of application sessions observed in the week of Feb 21 to 27, 2015. We discover that the main part of the traffic sessions is generated by a small fraction of applications and that 10 popular applications account for more than 80% sessions while all the other applications have less than 20% sessions. We believe that per-app traffic engineering is possible since we only need to track and monitor a few popular applications.

Figure 5 shows the real-time bandwidth monitoring of each application in the same week as that in Fig. 4. We

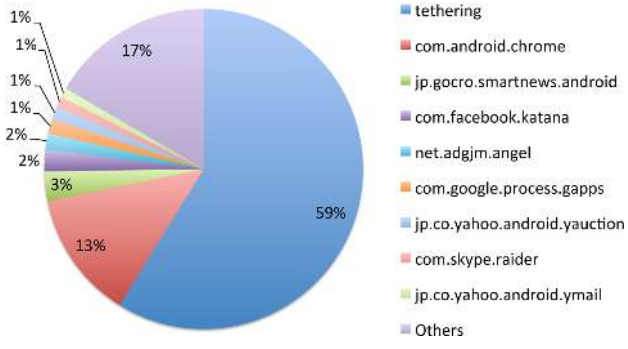


Fig. 4 Fractions of application sessions observed (Feb. 21–27, 2015).

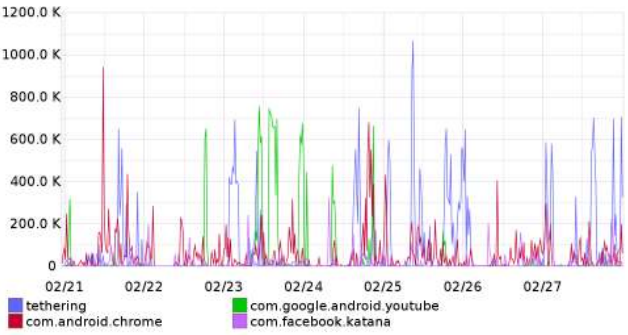


Fig. 5 Per-application bandwidth monitoring (Feb. 21–27, 2015).

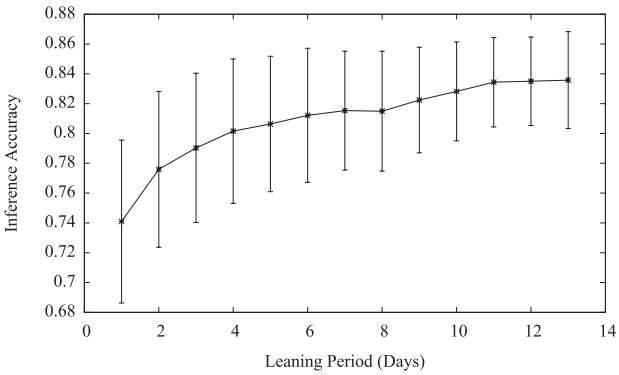


Fig. 6 Identification accuracy (without rejection region).

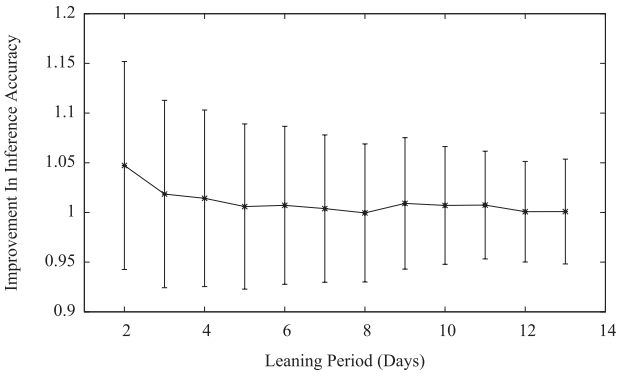


Fig. 7 Improvement in inference accuracy when extending learning period by one day.

can monitor not only traffic from the normal applications but also that from tethering devices. Usually, a tethering device can generate much more traffic than a smartphone. ISPs would want to make users pay extra charge for their tethering traffic on top of regular subscription plan. How to detect and block unauthorized tethering traffic has been an important issue for mobile carriers to manage their networks more efficiently. We should be able to provide the solution to this problem if an MVNO adopts our proposed mechanism of creating application/device specific slices.

4.4 Application Identification Without Customizing Smartphones

In order to examine the validity of our proposal introduced in Sect. 3.5, that is, a method for inferring applications from given traffic in real-time by machine learning using reliable training data generated by the customized smartphones, we have conducted analysis on the traffic data captured at a real MVNO deployed with FLARE programmable nodes.

We have distributed 65 customized smartphones to the university students and get them connected to the real MVNO network with a FLARE programmable node deployed at the packet gateway (P-gateway). We have captured the traffic at the FLARE node during the period from 2014/11/20 to 2014/12/13, where the total traffic includes 500 kinds of mobile applications and the amount of the data is about 6 GB per day.

Analyzing the captured data, we have evaluated how accurately we can infer the application that has transmitted the packets by observing the characteristics of the flow of the packets after we characterize the traffic using a simple machine learning technique called random forest [5]. For example, we use a combination of flow characteristics such as addresses, ports, packet lengths, jitter, TCP headers, etc. In our analysis, after collecting the data for a various learning period, we train the classifier using them and compare the accuracy of inference of applications.

4.4.1 Inference Accuracy by Naïve Method

First, using random forest, we can identify the traffic with the accuracy of about 80% after 5 days of machine learning. Also, we observe that the benefit is marginal even if we increase the learning period.

Figure 6 shows the accuracy of the application identification by our method. This result shows our method can identify about 80% mobile application when we set the term of the learning period 5 days. The error bar of each plot represents the standard deviation (likewise in all the other graphs from here on).

We examine the effectiveness of expanding the length of the learning period. Figure 7 shows the improvement in inference accuracy by extending learning period by one day after a given period (x-axis). For example, the improvement is about 5% if we extend the learning period from two days to three. Figure 7 indicates that the improvement by extending the learning period by one day after five days is less than 1%.

4.4.2 Inference Accuracy with Rejection Region

Second, after the first experiment, we figure that our classifier cannot infer the correct application when the many candidate applications have the almost same likelihood of inference, especially when we encounter a new application that has not been learned before. In this case, we should categorize that as “unknown” (meaning unable to classify) to avoid incorrect inference. Therefore, we set the rejection region and make the classifier reject the inference when it

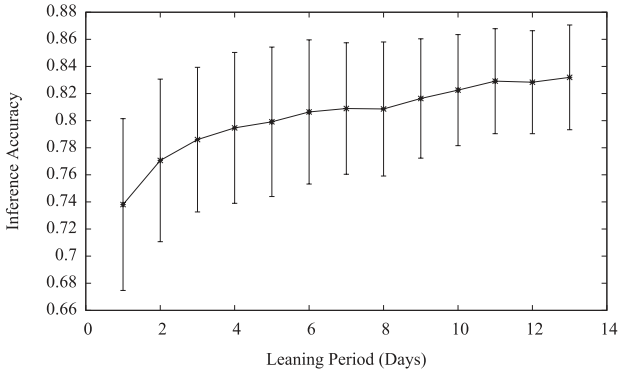


Fig. 8 Identification accuracy (with rejection region, including rejection).

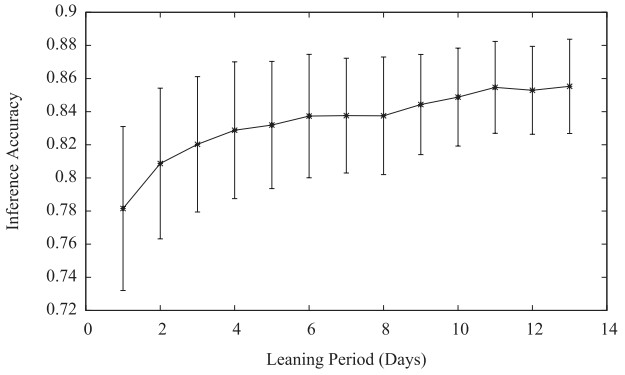


Fig. 9 Identification accuracy (with rejection region, excluding rejection).

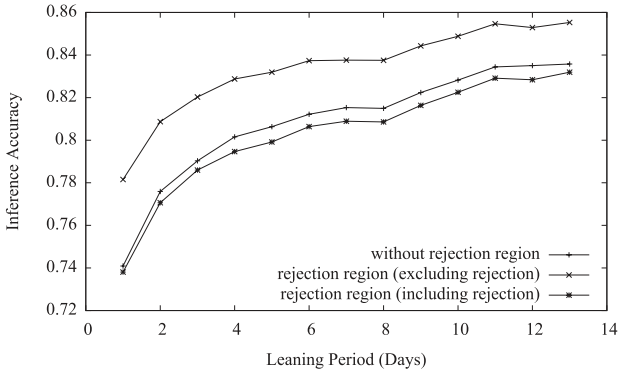


Fig. 10 Accuracy inference comparison among all methods.

cannot identify the application precisely.

Figures 8, 9 and 10 show the following results. Figure 8 shows the accuracy of all the inferences including the rejected ones, i.e., calculates the accuracy treating the rejected inferences as failure. Figure 9 shows the accuracy of all the inferences except the rejected ones. Figure 10 shows the comparison of the accuracies mentioned above without error bars for the sake of visibility.

These graphs indicate that setting the rejection region reduces false positive and that expanding the learning period reduces the variance of the accuracy and raises the mean

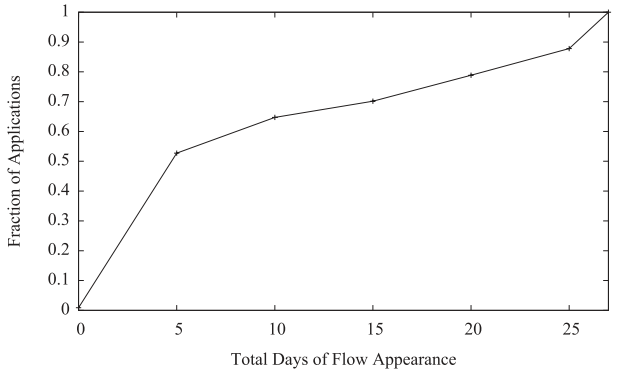


Fig. 11 Cumulative distribution of applications with total days of flow appearance.

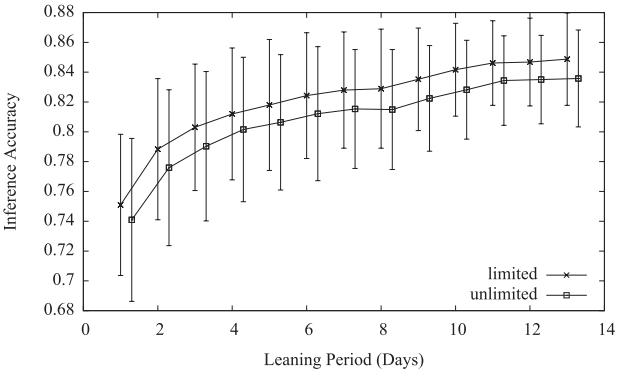


Fig. 12 Effect of limiting scope of inference (limited: limiting the number of target applications to those frequently observed, unlimited: inferring all the applications observed).

of the accuracy, thus, it is meaningful to set the rejection region.

4.4.3 Limiting the Scope of Inference

Figure 11 shows cumulative distribution of applications with total days of flow appearance, i.e., how many days we observe application flows during the course of the experiment. We observe that most applications do not always connect to the Internet and some of them are not observed all the time, which may have negative impact on the accuracy of inference.

Therefore, we evaluate the effect of limiting scope of inference, i.e., limiting the number of target applications to the ones that are frequently observed (more than 6 days in total) as shown in Fig. 12. Figure 12 indicates that if we limit the number of target applications to the ones that are observed more than 6 days during our experiment of 21 days, we can improve the accuracy of application inference by more than 2%.

5. Future Work

There are several possible improvements to be considered in near future in our proposed system.

First, in terms of application inference shown as one of the applications in Sect. 4.4, we plan to employ the other more advanced machine learning techniques. Our immediate future plan in this regard is to extend our analysis by using the other techniques such as support vector machine, bag of words, deep learning etc.

As for the future work for the proposed fundamental MVNO infrastructure, there are several important extensions we plan to explore.

First, we plan to design high-speed network functions within SDN data-plane using FLARE nodes so that we may execute a simple network function without external computational boxes beside FLARE nodes since usually the data center rack space cost is the primary concern for most MVNOs. To achieve high-speed packet I/O, an SDN data plane element built on top of FLARE must receive packets from physical link directly, bypassing Linux kernel network stack. This may speed up packet-level network functions such as router and load balancer to tens of millions packets per second. Session-level network functions such as Web HTTP proxies that perform stateful data processing with high-layer information are still beyond the current SDN architecture where only simple stateless actions are supported. We plan to integrate more high speed network functions such as user-space TCP/IP stack into SDN data plane without sacrificing the performance too much.

Second, our work can be extended to support non-IP data plane network functions such as information-centric network (ICN). In a software-defined ICN network, the notion of a flow can be mapped to an ICN entry at the edge network while the core ICN functions can be implemented in a non-IP environment.

Finally, our work can be extended to support intelligent network management. For example, with big-data analysis of social networks (e.g., Facebook and Twitter) and application/device based flow patterns collected in real-time, we can reactively deal with possible network disruptions caused by large social events, the Internet flaming acts, natural disasters, etc. We may also proactively program the data-plane for business agility and fast disaster recovery.

6. Conclusion

Our contributions in this paper are five-fold.

First, we propose *application and/or device specific slicing* applying the concept of software-defined data plane to defining new services for MVNOs. More specifically, we use software-defined deeply programmable data plane to handle trailer bits to attach a slice identifier, so that we can classify application and/or device specific traffic into slices and apply fine-grained quality of services (QoS). Most MVNOs of today provide low bandwidth at low price and thus, they are forced into fighting for the market with ever-lower-cost subscription plans. However, low flat-rate bandwidth services may not be attractive to users any more, since certain applications, such as a YouTube browser needs more bandwidth than low flat-rate bandwidth. Our solution can provide pay-as-you-go bandwidth services for a specific set of applications of customers' choice, and low flat-rate bandwidth services for the rest of applications. We expect our proposed system change business models of MVNOs and enhance the market of the MVNO business.

Second, we also introduce various applications of our proposed system, e.g., (1) traffic engineering based on application names, application processes, device types, and device status/location etc., (2) value-add services for specific applications and devices such as acceleration of content access and traffic reduction through compression/decompression and packet caching, (3) realizing in-network security such as malwares containment in a slice, and in-network parental control, (4) big-data analysis to improve the bandwidth utilization according to the statistical usage of applications and devices, and (5) application and device specific QoS even for unmodified smartphones, using machine learning on the training data collected from the customized smartphones to tag application/device information.

Third, our contribution includes not only providing new services for MVNOs, but also pointing out a compelling use case of software defined data plane, which is extended from the current SDN and NFV for allowing one (1) to define useful data processing within data plane in SDN and (2) to publish the access method to them as a (sub)set of southbound interface (SBI). We strongly believe that there are more and more useful use cases of software-defined data plane.

Fourth, we show our prototype implementation using FLARE deeply programmable nodes in a real MVNO network and proves that our concept in fact works in the real environment. We also show a preliminary analysis and evaluation using collected data to justify one of the applications, a new method of identifying applications from the traffic of unmodified smartphones by machine learning using the training data collected from the customized smartphones. We show that a simple machine learning technique such as random forest achieves about 80% of accuracy in application identification.

At last, another rather high-level contribution is, while most people are paying attention to OPEX/CAPEX reduction in SDN/NFV, we attempt to create new values out of applications of SDN/NFV. For this, we believe that it is important to think *application-driven programmable networking* where starting from the application that cannot be built without the help from the in-network functions, i.e., the network functions embedded inside the data plane of SDN solution. Developing generic infrastructure to accommodate all the applications, that is, bottom-up approach may not correctly define APIs. It is important to think top-down, from applications that do not exist today due to the limitation in the network, down to defining what are necessary inside the data plane of SDN.

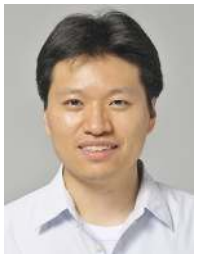
We strongly believe that enabling deeper programmability in SDN data-plane with ease of programming and reasonable performance surely open the door to bringing more

innovations.

References

[1] Flare: Deeply programmable network node architecture. http://netseminar.stanford.edu/10_18_12.html
[2] Geni engineering conference 20. http://groups.geni.net/geni/wiki/GEC20Agenda/EveningDemoSession
[3] Opendataplane. http://www.opendataplane.org
[4] Protocol oblivious forwarding. http://www.poforwarding.org
[5] L. Breiman, "Random forests," Mach. Learn., vol.45, no.1, pp.5–32, 2001.
[6] H. Farhady and A. Nakao, "TagFlow: Efficient flow classification in SDN," IEICE Trans. Commun., vol.E97-B, no.11, pp.2302–2310, Nov. 2014.
[7] M. Fukushima, Y. Yoshida, A. Tagami, S. Yamamoto, and A. Nakao, "Toy block networking: Easily deploying diverse network functions in programmable networks," Proc. 2014 IEEE 38th International Computer Software and Applications Conference Workshops, pp.61–66, 2014.
[8] A. Nakao, "Software-defined data plane enhancing SDN and NFV," IEICE Trans. Commun., vol.E98-B, no.1, pp.12–19, Jan. 2015.
[9] A. Nakao and P. Du, "Application and device specific slicing for MVNO," 2014 First International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), pp.1–5, Oct. 2014.
[10] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP fast open," Proc. Seventh Conference on Emerging Networking EXperiments and Technologies — CoNEXT'11, pp.1–12, 2011.

Akihiro Nakao received B.S. (1991) in Physics, M.E. (1994) in Information Engineering from the University of Tokyo. He was at IBM Yamato Laboratory, Tokyo Research Laboratory, and IBM Texas Austin from 1994 till 2005. He received M.S. (2001) and Ph.D. (2005) in Computer Science from Princeton University. He has been teaching as an associate professor (2005–2014) and as a professor (2014-present) in Applied Computer Science, at Interfaculty Initiative in Information Studies, Graduate School of Interdisciplinary Information Studies, the University.



Ping Du received B.E. and M.E. degree from University of Science and Technology of China in 2000 and 2003, respectively. He received a Ph.D. from the Graduate University for Advanced Studies in Japan in 2007. From 2008, he worked for the National Institute of Information and Communication Technologies (NICT) of Japan. Now, he works for the University of Tokyo as a project assistant professor. His research interests include optical network, network security, network virtualization etc.

Takamitsu Iwai received B.A. degree from University of Tokyo in 2015. He is a graduate student at Graduate School of Interdisciplinary Infomation Studies, Univesity of Tokyo now.