IEICE TRANS. COMMUN., VOL.E101–B, NO.7 JULY 2018

INVITED PAPER *Special Section on Communication Quality in Wireless Networks*

Toward In-Network Deep Machine Learning for Identifying Mobile Applications and Enabling Application Specific Network Slicing

Akihiro NAKAO[†a)] ***and*** **Ping DU**[†b)]**,** ***Members***

SUMMARY In this paper, we posit that, in future mobile network, network softwarization will be prevalent, and it becomes important to utilize deep machine learning within network to classify mobile traffic into fine grained slices, by identifying application types and devices so that we can apply Quality-of-Service (QoS) control, mobile edge/multi-access computing, and various network function per application and per device. This paper reports our initial attempt to apply deep machine learning for identifying application types from actual mobile network traffic captured from an MVNO, mobile virtual network operator and to design the system for classifying it to application specific slices.

key words: *software-defined networking (SDN), network functions virtualisation (NFV), network virtualization, 5G, network slicing*

1. Introduction

Network softwarization is an overall transformation trend for designing, implementing, deploying, managing and maintaining network equipment and network components by software programming, exploiting characteristics of software such as flexibility and rapidity of design, development and deployment throughout the life-cycle of network equipment and components [1], [2].

Network softwarization is considered a main driving factor for enabling network slicing [3], as realization of network functions and equipment by software program is considered a key to the network slicing concept. Software-defined networking (SDN) and network function virtualization (NFV) are the two main enabling technologies in creating network slices on physical infrastructure resources.

In the meantime, exploiting the flexibility in rapidly developing and deploying network functions driven by network softwarization has brought another interesting direction of research, "in-network deep machine learning", as reported by [4]–[6]. Since data plane, a.k.a., a set of traffic forwarding functions, is now software-defined, implementing deep machine learning within data plane becomes feasible. Deep Machine Learning [7]–[9] in this paper is defined as machine learning with deep neural networks composed of multiple processing layers to learn representations of data with multiple levels of abstraction. Each layer is a collection of connected computing units called "neurons" that can process input data and pass the output result to downstream neurons.

In this paper, we posit that, in future mobile network, network softwarization will be prevalent, and it becomes important to utilize deep machine learning within network to classify mobile traffic into fine grained slices, by identifying application types and devices so that we can apply Quality-of-Service (QoS) control, mobile edge/multi-access computing, and various network function per application and per device. This paper reports our initial attempt to apply deep machine learning for identifying application types from actual mobile network traffic captured from an MVNO, mobile virtual network operator and to design the system for classifying it to application specific slices.

Our contributions in this paper are as follows. First, we posit that in future mobile network, in-network deep machine learning for application and device specific identification and traffic classification becomes possible and show the ground for this position by our preliminary analysis.

Second, we propose "speculative data collection" techniques for in-network deep machine learning for application identification. The technique is elaborated in Sect. 6 but we believe this technique is applicable to many other machine learning for traffic analysis.

The rest of the paper is organized as follows. Section 2 introduces the concept of application specific end-to-end network slicing. Section 3 presents our proof-of-concept of end-to-end networking slicing in a real MVNO network. Sections 4 present our position for in-network deep machine learning for identifying mobile applications for enabling application specific end-to-end network slicing. Section 5 proposes our prototype system and reports our preliminary results. Section 6 discusses speculative data collection for in-network deep machine learning. Finally, Sect. 7 briefly concludes.

2. Application Specific End-to-End Network Slicing

Network slicing has attracted a great deal of attentions in 5G mobile networks. Before network slicing gains popularity among mobile networking researchers, we have defined "slice" as *an isolated set of programmable resources to implement network functions and application services through software programs* for accommodating individual network functions and application services within each slice without interfering with the other functions and services on the coexisting slices [10]. In fact, the concept of slice in networking

Manuscript received October 16, 2017.
Manuscript revised December 12, 2017.
Manuscript publicized January 22, 2018.

[†]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

DOI: 10.1587/transcom.2017CQI0002

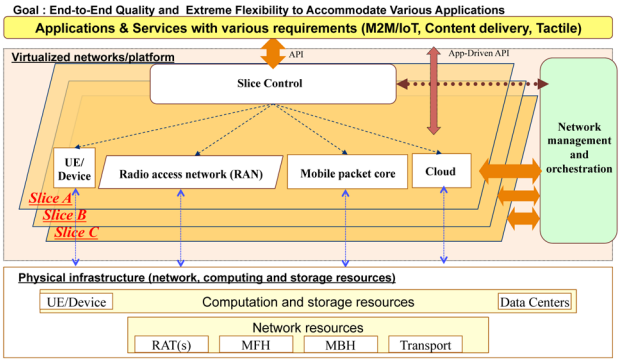


Fig. 1 End-to-end slicing concept [17].

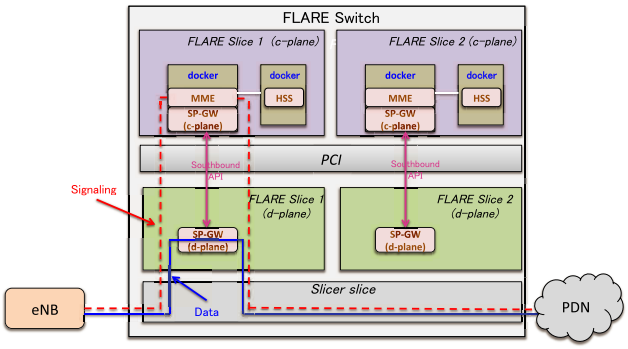


Fig. 2 Slicing EPC on FLARE switch [19].

has been introduced in network virtualization research efforts such as PlanetLab [11], in 2002, PlanetLab EU [12], GENI [13], VNode [14], FLARE [15], and Fed4Fire [16].

Thus, the slicing concept has been originally introduced in fixed networks, but it has come to make sense to extend to mobile networks as the target performance of 5G mobile networks becomes comparable to that of fixed ones. Therefore, 5GMF [17] network architecture committee led by the authors of this paper has identified the importance of end-to-end network slicing, all the way from user equipment (UE) to cloud data centers for enabling end-to-end quality and extreme flexibility to accommodate various applications, as shown in Fig. 1 included in the white paper [17]. The implications of the end-to-end network slicing architecture design shown in Fig. 1 have two significant aspects. First, using network, computing and storage resources embedded end-to-end, we should be able to program network functionalities all the way along end-to-end communication. Second, we can allocate an end-to-end network slice from UE to cloud data center *per application/service*.

The 5GMF white paper [17] considers one of the most important concepts to realize 5G mobile network is "extreme flexibility" partly enabled by network softwarization and network slicing. Since 5G mobile networks are expected to accommodate at least three, very different types of applications eMBB (enhanced MobileBroad Band), mMTC (massive Machine Type Communications), and URLLC (Ultra Reliable and Low Latency Communications), it makes sense for 5G mobile networking infrastructure to support very different QoS and computations for each type of communications, thus, it is a crucial requirement for the infrastructure to support extreme flexibility in 5G mobile networks.

3. Prototyping Application Specific End-to-End Slicing

In this section, we develop a proof-of-concept of end-to-end network slicing in a real MVNO mobile network, where we introduce slicing smart phones with trailer-slicing technology, RAN slicing with MVNO, multi-access computing and Evolved Packet Core (EPC) slicing on FLARE to classify application specific traffic into slices and apply different services to different mobile applications.

3.1 Trailer-Based UE Slicing

It is too power-consumptive to run hypervisors (e.g., KVM) or even container-based visualization (e.g., LXC) based slicing technologies on smartphones. Here we develop a much lighter UE slicing technologies, where we install our software on smartphones to capture the very first packets for an application and examine the process table and the socket table of the operating system to look for a corresponding application process name that uses the flow space and attach the information as a trailer. After adjusting of header fields in Layer-3 and Layer-4, we can get packets with trailers through existing network appliances since trailers are treated as Layer-7 data bits.

3.2 Application Specific Multi-Access Computing Slicing

In our prototype, we classify traffic from phones to different virtual network functions accordingly. The traffic of different slices is isolated using VLANs on the hosting multi-access computing running Open vSwitch [18]. Packets from smartphones are classified and tagged with different `VLAN IDs` according to applications. In each slice of multi-access computing, we apply specific optimization policy according to applications. For example, we run HTTP caching service for web browsing (e.g., `Chrome`), video transcoding service for video streaming (e.g., `YouTube`), and bandwidth control for `Tethering` traffic.

3.3 EPC Slicing on FLARE

In this section, we introduce how to implement an EPC slice in a FLARE slice shown in Fig. 2, where signaling related EPC entities (e.g., MME) are implemented in control plane while user data forwarding and processing (e.g., S-GW and P-GW) are implemented in data plane.

3.3.1 Data Plane

We offload the GTP-U channel creation and user data processing from control plane to data plane, which is imple-

mented with GTPv1-U kernel module in the original OAI software. In order to scale network functions, we implement SP-GW data plane components with chained Click elements. When a FLARE node receives packets from eNB, its classifier called *Slice slicer* will classify packets to different slices as well as classifying signaling packets, e.g., GTP-C from data packets, e.g., GTP-U. The signaling packets are forwarded to control plane while the data packets are processed in data plane with many-core processors.

3.3.2 Control Plane

We execute the signaling entities of an EPC slice, e.g., MME and the control plane of SP-GW in a Docker instance. We also execute HSS entity in another Docker instance within the same *sliver* of a FLARE node. A sliver is defined as execution environment in a FLARE node that is part of a network wide slice constructed across multiple FLARE nodes.

These two Docker instances are isolated without interfering with each other. For example, we can install different versions of software packages in MME and HSS instances even if they may conflict with each other when installed on the same host machine.

The interfaces between EPC and HSS entities are implemented with internal Ethernet links. They can communicate with each other via TCP and SCTP protocols.

4. In-Network Deep Machine Learning for Identifying Mobile Applications

4.1 Application Identification

In order to enable application specific network slicing, the first challenge we face is to classify traffic by applications.

Conventional application identification [20] relies on packet headers (destination IP and/or port), or signatures reconstructed from payloads. Traditionally, many applications use 'well-known' ports on their dedicated servers. Recently, the packet header based traffic identification [21] fails due to the fact that many mobile applications use common ports (e.g., 80 and 443) and are often hosted on some public clouds, which makes it hard to classify applications only by destination ports in packet headers.

The signature-based application identifications [22], [23] impose significant complexity and processing load on the device, which must be frequently updated to keep up with the latest knowledge of application protocol semantics. When an application specification changes or a new application appears, people must start over for exploring valuable signatures, which is time-consuming and labor-intensive.

Usually, it is not obvious to identify an application that has transmitted a given flow that network equipment observes, as network infrastructure, for the simplicity's sake, has been designed to be independent of application contexts. In general, network stacks below Layer 7 (L7) should not (or must not) care about application contexts as applications are designed to run at end systems and part of applications are not to be executed within networks.

4.2 Application Specific Data Processing

Network operators are interested not only in identifying and classifying mobile traffic for different QoS, but also in applying specific optimization policy according to applications. For example, we run HTTP caching service for web browsing (e.g., Chrome), video transcoding service for video streaming (e.g., YouTube), and bandwidth control for tethering traffic.

4.3 In-Network Deep Machine Learning

There are several ways to achieve application identification and classification, e.g., packet header marking such as Type of Service (ToS) and its extended version, and deep packet inspection (DPI) where we detect so called signature per application, application specific characteristics learned from packet payloads, as well as packet headers.

DPI is often implemented in middle boxes such as firewalls, but it is mainly used for detecting attacks and embedded malwares. Application identification and classification for application specific network slicing is much more challenging as we must be able to classify generic applications, not just malicious ones, dealing with a broader scope of applications. Also application specific information is most likely encrypted so it becomes harder and harder to identify application from the content conveyed in packet payload.

As network softwarization becomes popular, it is natural to embed much more intelligent network functions within network. Especially when data plane is software-defined, it is feasible to implement much more powerful machine learning methods than just simple pattern matching to learn application specific characteristics to enable application identification.

Machine learning based identification has been proposed in [24], [25]. The selected features are fed into some kind of classifier such as Naive Bayes [26], K-Means [27] and Neural Network [28]. Conventional machine learning techniques are limited in processing natural data in their raw form. Usually, much expertise is required to construct pattern-recognition feature vectors from raw data.

Deep learning has enabled many practical applications of machine learning, such as image classification and object detection. The most important advantage of deep learning over conventional machine learning is replacing handcrafted features with efficient algorithms for feature learning and hierarchical feature extraction. A deep learning system consisting of many layers of neurons can be fed with raw data, to increasingly abstract the raw data, and automatically discover the representations needed for detection or classification.

We posit that network softwarization opens a door to in-network deep machine learning for identifying and classifying mobile applications and for enabling application network slicing.

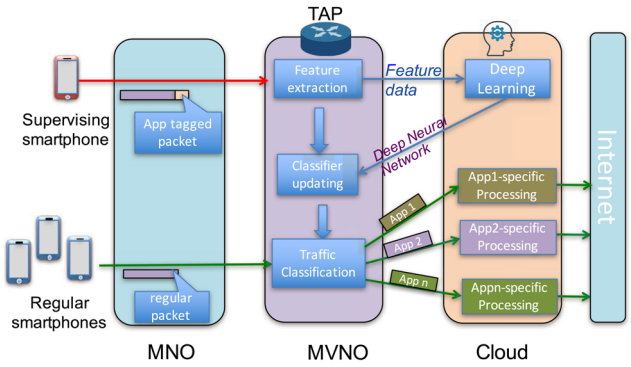


Fig. 3 In-network deep learning-based application classification and processing.

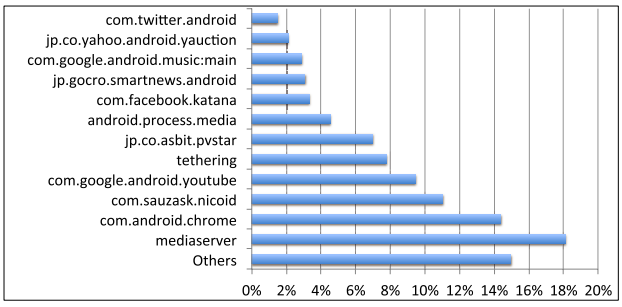


Fig. 4 Distributions of traffic volume by popular applications.

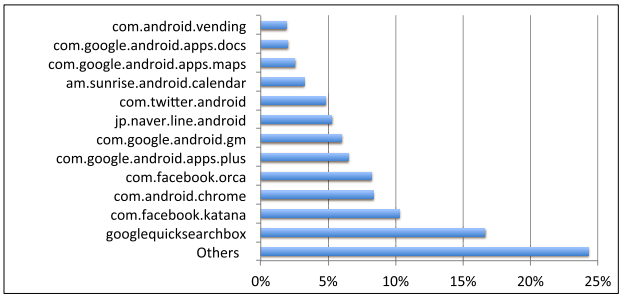


Fig. 5 Distributions of user interaction time by popular applications.

5. Prototype of In-Network Deep Learning

5.1 Real-Time Application Classification System

As shown in Fig. 3, we propose a method for identifying applications from given traffic in real-time by deep machine learning using reliable training data generated by supervising smartphones. We use a small number of customized smartphones as supervising smartphones to generate training data where packets are tagged with the information of the application transmitting them. Then we apply deep learning on given flow and extract the useful features in a train of packets contained in the flow, without looking into the payload of packets.

Compared with conventional work on identifying applications from the traffic trace relies on DPI of the user data, our method has two benefits: (1) we don't need to inspect packet payload of both training and test data so that we may not risk privacy violation, and (2) our training data are generated in real-time with low-cost at 100% accuracy because of packet tagging even when the packet payload data is encrypted.

5.2 Preliminary Study: Analysis of Mobile Traffic

We have distributed over 60 smartphones to the volunteers (including university students/staffs, company/government employees, and media people) and get them connected to the MVNO experimental network. We have been capturing the traffic at the FLARE node for more than 500 days since 2014/11/20, where the total traffic includes over 500 kinds of mobile applications and the total amount of the collected data is more than 2.1 terabytes.

We first analyze the traffic characteristics of different mobile applications. We believe that detailed study of MVNO network not only help the current MVNOs improve their services but also benefit the 5G research and development in terms of network slicing, especially for allocating isolated resources for different applications and services.

Figures 4 and 5 show the distributions of traffic volume and user interaction time by popular applications separately [18]. We observe that about 43% of traffic volume is occupied by video streaming applications although only 5% of user time is spent on them.

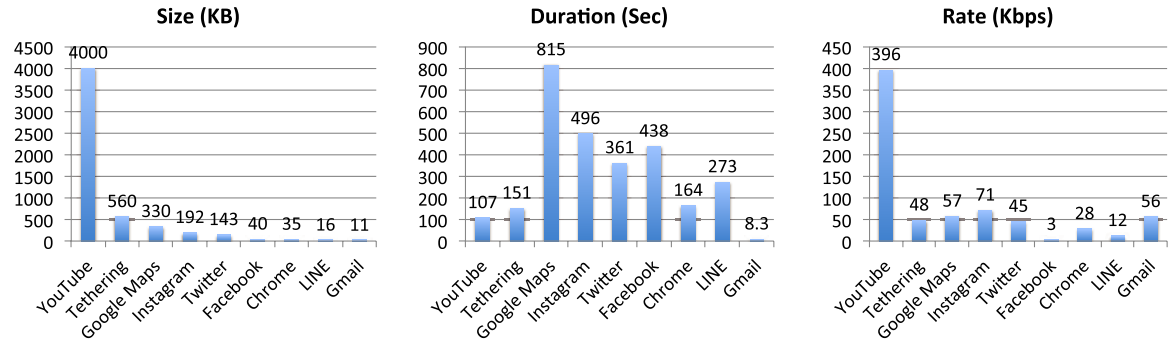
As a comparison, about 22% of user time is spent on social network applications, but only 4.8% of traffic volume is occupied on them. Web browsing applications occupy about 14% traffic volume and 22% of user time.

Besides video streaming applications that account for a large fraction of traffic volume in the long-term observation, we also observe that `Tethering` traffic dominates in certain days although `Tethering` occupies 7.8% of traffic volume in the long-term observation.

`Tethering` allows a smartphone to share its cellular (e.g., 3G/LTE) data connection with other devices using a WiFi hotspot or direct cable connection. Usually, a tethering device can generate much more traffic than a smartphone. ISPs would like to devise different charging schemes for their tethering traffic on top of the regular subscription cost.

Figure 6 shows the traffic patterns (average TCP flow size, average flow duration, and average flow rate) of some popular applications. The average flow rate is calculated with normalized flow duration. The flow rate of `YouTube` traffic (396Kbps) is much higher than the other applications (<100Kbps). We observe that the average rate of each `Tethering` flow is only about 48Kbps, almost the same as those of other mobile applications.

Surprisingly, the flow rate of `Facebook` is only 3Kbps, which is even smaller than that of `Gmail`. This is because `Facebook` application tends to hold persistent TCP sessions to transmit multiple text or media objects while `Gmail` application prefers to terminate TCP sessions when idle.



(a) (b) (c)

Fig. 6 Flow characteristics of popular applications.

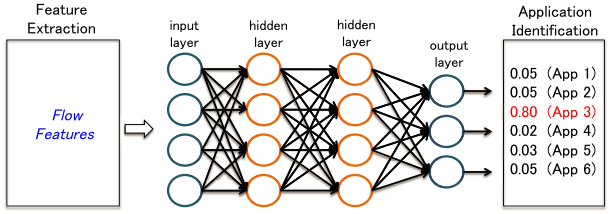


Fig. 7 An example of application identification with deep neural networks with extracted features.

These results give us insight into tackling the problem of application identification by deep machine learning. There are popular applications with a large volume of data exchanged, but on the other hand, the other application traffic may not be significant in volume. It is probably hard to achieve machine learning using a small amount of sample traffic.

5.3 Application Identification with Deep Neural Network

As shown in Fig. 7, our training model is defined based on deep neutral networks (DNN) with an input layer, multiple fully connected hidden layers and an output layer. Each layer is a feed-forward neural network.

The input layer provides training data from outside and feed the training data $\mathtt{X}$ into the training network. There is no computation in the input layer, which just passes the received information to hidden layer.

Each hidden layer consists of (1) taking the previous layer output $\hat{\mathtt{Y}}$ as input $\mathtt{X}$ and having weight matrices $\mathtt{W}$ and bias vector $\mathtt{b}$ associated to those inputs ($\hat{\mathtt{Y}} = \mathtt{W} \cdot \mathtt{X} + \mathtt{b}$), (2) passing the responses through a rectified linear function ($relu(\hat{\mathtt{Y}}) = max(\hat{\mathtt{Y}}, 0)$)

The output layer is built with a softmax regression mode and its output is a probability vector $\hat{\mathtt{Y}}$ over the top $\mathtt{M}$ (e.g., 200 in our experiment) popular applications.

We train these models using a large set of N tagged flows $\{\mathtt{X}, \mathtt{Y}\}$, where label $\mathtt{Y}$ is one-hot vector indicating the true application. A cross-entropy loss function is used to compare the target $\mathtt{Y}$ and the softmax activation function applied to the model's prediction $\hat{\mathtt{Y}}$. The parameters of the network (weight matrices in the fully-connected layers and biases) are trained by back-propagating the derivative of the loss with respect to the parameters throughout the network, and updating the parameters via stochastic gradient descent.

Basically, training a network is to guide the parameters to the point that the softmax loss is minimized, which means to maximize the corresponding column of the true class in the output vector. Each hidden layer is also told to change their parameters so that their contribution to the prediction error is minimized. Therefore, each hidden layer is also an optimized classifier in its own plane.

5.4 Flow Features Extraction with Deep Neural Network

On the other hand, our training model also works as a feature extractor because the output value is discriminative to not only network parameters but also input features. Usually, a feature with big relevance to the output is considered as a useful feature.

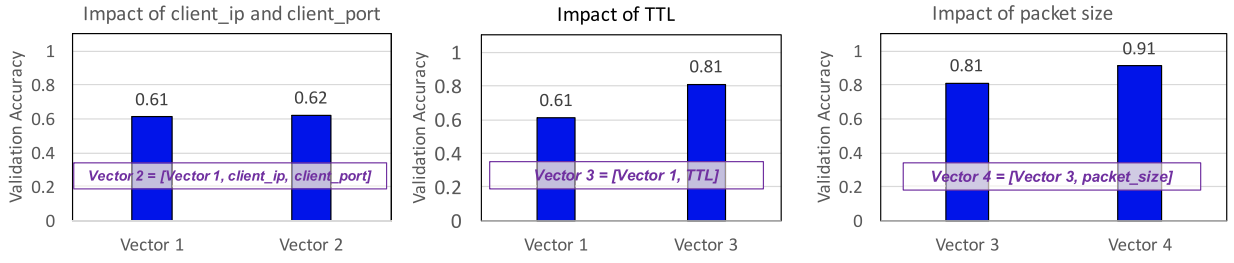
It is straightforward that `server_ip`, `server_port`, and `proto` could be the features to identify applications.

We set feature `Vector 1` = [`server_ip`, `server_port`, `proto`] as the base feature vector and try to find other useful features with our training model. In our training process, besides a set of traffic data as training data, we also set another small set of traffic data as validation data to validate the training model during training process.

As shown in Fig. 8(a), we first check the impact of [`client_ip`, `client_port`]. Our finding is that there is no obvious appearance pattern in [`client_ip`, `client_port`] that affects the validation accuracy. So we take [`client_ip`, `client_port`] as not useful patterns in future train model.

In Fig. 8(b), we set feature `Vector 3` = [`Vector 1`, `TTL`] and check impact of `TTL`. We find `TTL` is in fact useful in application identification. We believe this is because `TTL` is a metric of the distance from the application server to the FLARE node located near P-Gateway, which is highly dependent on application type. Then we add `TTL` to our feature vector and check the impact of packet size in Fig. 8(c).

We also find that packet size is a useful feature. This is because client and server need to exchange information during connection establishment. The size of exchange in-



(a) (b) (c)

Fig. 8 Impact of features in application identification.

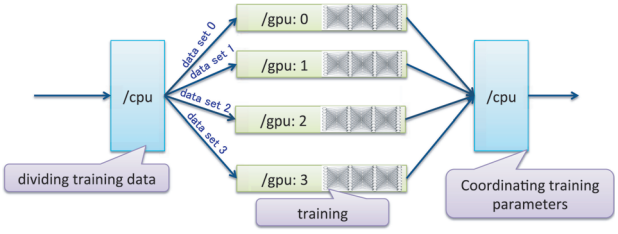


Fig. 9 Architecture of multi-GPU deep neural networks.

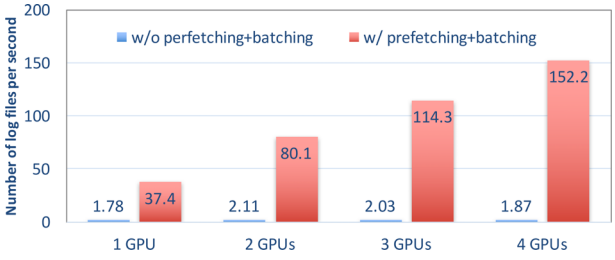


Fig. 10 Training performance of multi-GPU system.

formation is usually application specific. Therefore, we add sizes of the first few packets of each flow as a useful feature to our training model.

5.5 Multi-GPU Acceleration Framework

As shown in Fig. 9, we begin with a brief introduction of our multi-GPU DNN platform, consisting of one Intel 8-core Xeon E5-2670 2.60 GHz processor and four NVIDIA GTX1080 cards, each of which has 8GB GDDR5X memory and 20 Streaming Multiprocessors (SMs). Each SM contains 128 CUDA cores, resulting in 2560 CUDA cores per GPU in total. The processing power of GPU comes from its hundreds of cores.

According to our test, at the peak performance one GTX1080 GPU is comparable to about ten E5-2670 processors. So we offload all training tasks to GPUs while use CPU to preprocess and slice training data into GPUs and coordinate training parameters during training.

A multi-GPU system has two performance issues: (i) whether GPU is fully utilized and (ii) linear performance scalability with additional cores. After careful profiling, our finding is that GPUs can be starved for data due to the shortage of the bandwidth from main memory to GPU memory.

To address these issues, we propose two optimization strategies: *batching* and *prefetching*. With *batching*, we can combine the training data from a set of small log files into a large file and transfer it from main memory to GPU as a unit. With *prefetching*, we can preload training data into GPU as much as possible so that GPU can reduce the starving time during training.

Figure 10 compares the performance of training speed when without or with batching and prefetching. The test results show that after we apply batching and prefetching, not only the training performance of single GPU is speeded up obviously but also the performance of multi-GPU training system is linear to the number of GPUs.

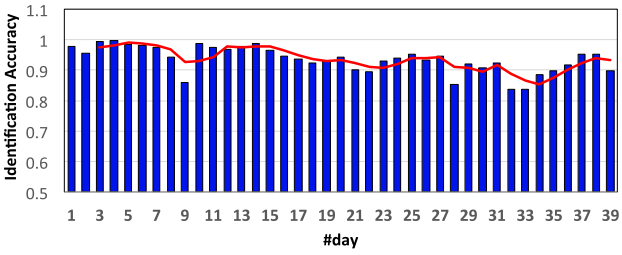


Fig. 11 Accuracy of App Identification over 39-day traffic [avg. 93.5%].

5.6 Preliminary Experimental Results

We use two-week MVNO data as training data, where each day of traffic consists of about 40000 flows. Besides training and test, we use one-day of traffic as validation in training. The distribution of 200 applications is shown in Fig. 4 and Fig. 5.

As shown in Fig. 11, by using a tuple of <`dst_ip`, `dst_port`, `protocol`, `ttl`, `packet_size`> as the features of application traffic captured at an MVNO, we can successfully identify 200 mobile applications with about 93.5% accuracy over 39-day traffic using a 8-layer Deep Neural Network with TensorFlow [29], where each hidden layer consists of 40000 (200x200) neurons.

An immediate question is how to improve the accuracy of application identification. According to our preliminary experimental results, an efficient way to improve the performance is to increase the volume of training data. Figure 12 shows the effect of increase in data volume. The test results

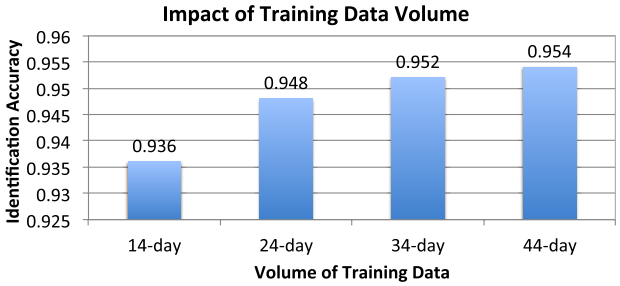


Fig. 12 Impact of training data volume in application identification.

show that the accuracy is increasing gradually with the volume of training data. However, we believe further study is necessary, as this is only our first endeavor to improve the accuracy of application identification.

6. Speculative Data Collection

In this section, we discuss a method called "speculative data collection" we have defined as one of the important challenges for in-network deep machine learning. We have discovered its importance during the course of conducting our experiments described in previous sections.

Traditionally, in most network data science research efforts, we passively collect all the traffic data and analyze them later in large computation facilities, so called, big data analysis, or with limited computational resources perform random sampling to reduce the size of the volume.

However, for a certain research problem, such as application identification, the diversity of data is more useful than its volume. In fact, the analysis of duplicated, similar data often incurs additional cost for computation without adding much value to the accuracy, especially for deep neural network processing.

Therefore, we need intelligent sampling of data to be analyzed for this type of data analysis. That is, if data source is not adding much value to the accuracy, we may not want to use the data source producing such kind of data, but rather choose the data source producing diverse data.

For our project described in this paper, it is useful to speculatively identify smartphones producing a wide variety of application data, especially constantly installing newly emerging applications that will possibly gain popularity.

As shown in Fig. 13, in our proposed method of in-network deep machine learning for application identification, we selectively annotate data from data sources that are considered useful for generating supervising data in real-time. Since we perform online supervising data collection and training at the same time, we can adapt to the changes in data characteristics, thus minimizing the turn around time between data collection and application of the result of machine learning.

We currently manually select data sources for achieving speculative data collection. However, in future, it would be desirable to perform it automatically, possibly by another level of machine learning.

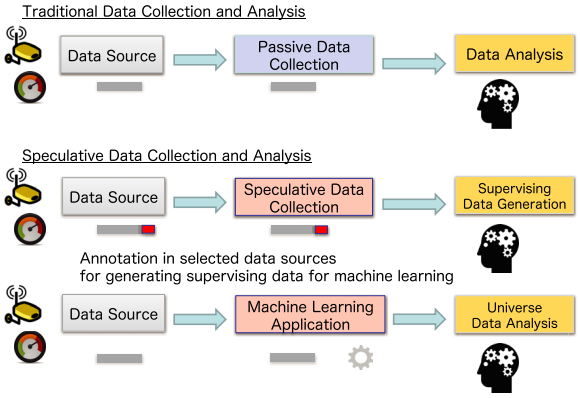


Fig. 13 Speculative data collection.

7. Conclusion

In this paper, we posit that, in future mobile network, network softwarization will be prevalent, and it becomes important to utilize deep machine learning within network to classify mobile traffic into fine grained slices, by identifying application types and devices so that we can apply QoS control, mobile edge/multi-access computing, and various network function per application and per device. This paper reports our initial attempt to apply deep machine learning for identifying application types from actual mobile network traffic captured from an MVNO, mobile virtual network operator and to design the system for classifying it to application specific slices.

References

[1] "Focus group of imt-2020," http://www.itu.int/en/ITU-T/focusgroups/imt-2020/Pages/default.aspx, 2015.
[2] "Fg imt-2020: Report on gap analysis," http://www.itu.int/en/ITU-T/focusgroups/imt-2020/Documents/T13-SG13-151130-TD-PLEN-0208%21%21MSW-E.docx, 2015.
[3] A. Nakao, P. Du, Y. Kiriha, F. Granelli, A.A. Gebremariam, T. Taleb, and M. Bagaa, "End-to-end network slicing for 5G mobile networks," IPSJ Journal of Information Processing, vol.25, pp.153–163, 2017.
[4] A. Nakao, "App-specific edge computing and in-network deep learning," Special Session, The 19th Asia-Pacific Network Operations and Management Symposium, Special Section 2, APNOMS, 2017.
[5] A. Nakao, "Network softwarization and in-network deep learning," Keynote, The 7th Symposium on Network Virtualization 2017.
[6] A. Nakao, "Network softwarization and in-network deep learning," Keynote, The 12th International Conference on Future Internet Technologies, CFI, 2017.
[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol.521, no.7553, pp.436–444, 2015.
[8] I.H. Witten, E. Frank, M.A. Hall, and C.J. Pal, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, 2016.
[9] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol.61, pp.85–117, 2015.
[10] A. Nakao, "Network virtualization as foundation for enabling new network architectures and applications," IEICE Trans. Commun., vol.E93-B, no.3, pp.454–457, March 2010.
[11] "Planetlab," http://www.planet-lab.org, 2012.
[12] "Planetlab," https://www.planet-lab.eu, 2012.

[13] "Geni," https://www.geni.net, 2007.
[14] K. Yamada, Y. Kanada, K. Amemiya, A. Nakao, and Y. Saida, "Vnode infrastucture enhancement — Deeply programmable network virtualization," 2015 21st Asia-Pacific Conference on Communications (APCC), IEEE, pp.244–249, 2015.
[15] "Flare: Open deeply programmable network node architecture," http://netseminar.stanford.edu/seminars/10_18_12.pdf
[16] "Fed4fire," https://www.fed4fire.eu
[17] 5GMF White Paper, "5G mobile communictions systems for 2020 and beyond," http://5gmf.jp/wp/wp-content/uploads/2016/09/5GMF_WP101_All.pdf, July 2016.
[18] P. Du and A. Nakao, "Application specific mobile edge computing through network softwarization," IEEE International Conference on Cloud Networking (CloudNet), 2016.
[19] A. Nakao, P. Du, Y. Kiriha, F. Granelli, A.A. Gebremariam, T. Taleb, and M. Bagaa, "End-to-end network slicing for 5G mobile networks," Journal of Information Processing, vol.25, pp.153–163, 2017.
[20] A. Callado, C. Kamienski, G. Szabó, B.P. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," IEEE Commun. Surveys Tuts., vol.11, no.3, pp.37–52, 2009.
[21] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry," RFC 6335, Aug. 2011.
[22] J. Van Der Merwe, R. Caceres, Y.-H. Chu, and C. Sreenan, "mmdump: A tool for monitoring internet multimedia traffic," ACM SIGCOMM Comput. Commun. Rev., vol.30, no.5, pp.48–59, 2000.
[23] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of P2P traffic using application signatures," Proc. 13th international conference on World Wide Web. ACM, pp.512–521, 2004.
[24] J. Erman, A. Mahanti, and M. Arlitt, "Qrp05-4: Internet traffic identification using machine learning," Global Telecommunications Conference, 2006. GLOBECOM'06, IEEE, pp.1–6, IEEE, 2006.
[25] S. Zander, T. Nguyen, and G. Armitage, "Self-learning IP traffic classification based on statistical flow characteristics," Passive and Active Network Measurement, pp.325–328, 2005.
[26] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: When randomness plays with you," ACM SIGCOMM Comput. Commun. Rev., vol.37, no.4, pp.37–48, 2007.
[27] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core," Proc. 16th international conference on World Wide Web, pp.883–892, ACM, 2007.
[28] T. Auld, A.W. Moore, and S.F. Gull, "Bayesian neural networks for internet traffic classification," IEEE Trans. Neural Netw., vol.18, no.1, pp.223–239, 2007.
[29] "Tensorflow," https://www.tensorflow.org/

Akihiro Nakao received his B.S in Physics and M.E in Information Engineering from the University of Tokyo. He worked at IBM Yamato Laboratory, Tokyo Research Laboratory, and IBM Texas Austin. He received his M.S and Ph.D. in Computer Science from Princeton University. Since 2005, he has been an Associate Professor and is now a Professor in Applied Computer Science at the Interfaculty Initiative in Information (III) Studies, Graduate School of Interdisciplinary Information Studies, University of Tokyo. He has been appointed as Chairperson of Department in III. He has also been appointed Chairman of the 5G Mobile Network Promotion Forum (5GMF) Network Architecture Committee by Japanese government.

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 lecture. His research interests include optical network, network security, network virtualization, mobile network etc.