Received July 26, 2021, accepted September 5, 2021, date of publication September 13, 2021, date of current version September 22, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3112210

Intelligent Network Slicing With Edge Computing for Internet of Vehicles

PING DU[1], (Member, IEEE), AKIHIRO NAKAO[2], (Member, IEEE), LEI ZHONG[3], AND RYOKICHI ONISHI[3]
[1]Interfaculty Initiative in Information Studies, The University of Tokyo, Tokyo 113-0033, Japan
[2]Faculty of Engineering, The University of Tokyo, Tokyo 113-0033, Japan
[3]Toyota Motor Corporation, Tokyo 100-0004, Japan

Corresponding author: Ping Du (duping@g.ecc.u-tokyo.ac.jp)

This work was supported in part by Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant JP18K11255 and Grant 21K11843, and in part by a project commissioned by New Energy and Industrial Technology Development Organization (NEDO) in Japan.

ABSTRACT In this paper, we present an application-specific Multi-Access Edge Computing (MEC) network architecture by leveraging the Control and User Plane Separation (CUPS) in mobile core networks to offload data processing from central servers to edge servers to reduce the transmitted traffic volume and also the response latency of connected vehicle mobility service. We first apply deep learning to classify packets of different applications to different Radio Access Networks (RAN) slices for application-specific spectrum scheduling. Then, we slice Evolved Packet Core (EPC) and deploy EPC data plane slices on-demand for each application and route packets from RAN slices to edge servers. By applying network slicing, multiple RAN, EPC and MEC slices that support different categories of services with different quality of service (QoS) requirements can be deployed in the same physical infrastructure. We prototype the proposed application-specific CUPS architecture using modified open source software OpenAirInterface on our deeply programmable platform. The preliminary experimental results show the feasibility and efficiency of proposed application-specific CUPS architecture, which can achieve a significant decrease in transmission data volume and latency.

INDEX TERMS Edge computing, software-defined network (SDN), machine learning (ML).

I. INTRODUCTION

The evolving fifth Generation of Mobile Communications System (5G) communications are envisioned to be classified into three categories [1]–[3]: enhanced Mobile Broad Band (eMBB) to deliver gigabytes of bandwidth to mobile devices on-demand, massive Machine Type Communications (mMTC) to connect sensors and machines, and Ultra Reliable and Low Latency Communications (URLLC) targeted to low latency and reliable applications like autonomous driving.

Multi-Access Edge Computing (MEC) has been considered as an indispensable component for the 5G networks. It brings the applications from the centralized data centers to the network edge that is close to Radio Access Network (RAN) and User Equipments (UE). To meet the diverse QoS requirements from diverse kinds of applications, it is desirable to transfer traffic to dedicated edge servers to reduce the transmitted traffic volume and also the response latency of latency-critical applications.

However, how to effectively identify and classify applications in real-time is still an open issue especially in the RAN research area. As long as data has been transmitted from UEs into a mobile network, the contextual information of the data (e.g., which application the data belongs to and which device the data generated from) is hidden from the network alliances. Conventionally, there are several ways to achieve application identification and classification, e.g., packet header marking [4], and deep packet inspection (DPI) [5] to detect signature per application from packet payloads. But packet header marking fails to identify a broad scope of applications while DPI is becoming harder and harder due to that application-specific information conveyed in the payload is most likely encrypted.

To address this issue, we first propose an application-specific mobile network architecture utilizing in-network deep learning so that we can apply application-specific radio spectrum scheduling in RAN, Quality-of-Service (QoS) control and various network functions per application in core

The associate editor coordinating the review of this manuscript and approving it for publication was Celimuge Wu.

networks. In our design, 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 can apply in-network deep learning at Packet Gateway (P-GW) to identify mobile applications of other phones and classify applications to different virtual network functions for application-specific in-network processing (e.g., HTTP caching service for web browsing, video transcoding service for video streaming etc.).

We also attach the identification results to the downlink packets at the P-GW and transmit them to eNodeB (eNB). The eNB can apply application-specific spectrum scheduling based on the attached application information. For each UE, we set up one radio bearer and multiple S1 bearers between eNodeB (eNB) and EPC user planes. For a packet from different kinds of applications, eNB not only can assign different RAN resources to it but also can divert it to different user planes via different S1 bearers.

Then, we propose a deep learning-based application-specific control and user plane separation (CUPS) edge computing architecture to offload localized data processing to the local edge server to reduce the overload of the central server. By applying C/U plane separation, the Evolved Packet Core (EPC) control plane could be virtualized and located in a cloud environment while the user plane remains in the transport network. Moreover, we can deploy multiple user planes on-demand for a single control plane. As a result, CUPS supports the increase of data traffic by enabling to add more user plane nodes without changing the number of control plane nodes and reduces latency on application service by placing gateways as close as possible from the radio access networks or users. Therefore, CUPS is one of the indispensable technologies to enable edge computing and achieve the QoE anticipated with the 5G introduction.

Our contributions in this paper are as follows. First, we present our design of an in-network deep learning-based CUPS architecture for mobile edge computing. We present our design of an in-network deep learning-based mobile network. One advantage of the proposed architecture is that we only need to apply deep learning-based application identification once at packet gateway (P-GW) while the eNB can reuse the identification results piggyback on the packets from P-GW to RAN.

Second, we utilize in-network deep learning to identify and classify traffic and apply application-specific radio spectrum scheduling in RAN, Quality-of-Service (QoS) control, and various network functions. As far as we know, there are very few real implementations of application-specific spectrum scheduling although there are many simulation-based works [6], [7] on how to schedule ratio resources with different QoS requirements (e.g., real-time and delay-tolerant).

Third, we introduce our effort on prototyping the CUPS architecture. By flexibly deploying user plane nodes, we can offload the localized data processing at the local edge server instead of concentrating all the information at the central server. This mitigates the burden of both networks and the central cloud. There are several advantages of our prototype: (i) *Feasibility:* as far as we know, our work is one of the very few real implementations; (ii) *Scalability:* in our system, multiple user planes can be deployed on-demand; (iii) *Extensibility:* our work can be easily extended to work as one of the user plane functions (UPF) in 5G networks.

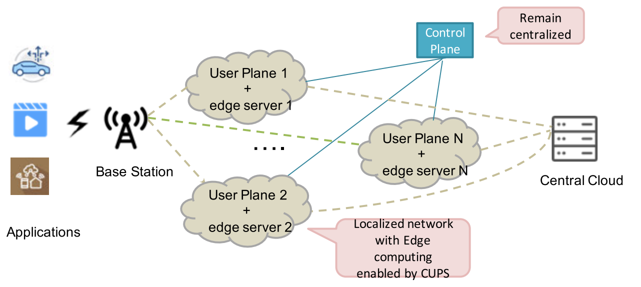

Remain centralized
Control Plane
User Plane 1 + edge server 1
User Plane N + edge server N
User Plane 2 + edge server 2
Base Station
Applications
Central Cloud
Localized network with Edge computing enabled by CUPS

FIGURE 1. **Architecture of edge computing with CUPS.**

The rest of the paper is organized as follows. Section II introduces the CUPS-based edge computing deployment. Section III introduces the design of in-network deep learning-based application-specific CUPS. Section IV presents our implementation of application-specific CUPS architecture. Section V reports some preliminary evaluation of our prototype system. Section VI briefly concludes and introduces future work.

II. EDGE COMPUTING WITH CUPS

In a 5G network, massive machine type communication (MTC), including narrowband (NB)-IoT has been approved by 3GPP, and it is intended to connect a massive number of small low-power sensor devices. Still, the data volumes are considered fairly modest. But adding to this, the current trend of concentrating data processing at central locations will cause huge data transmission traffic, which will lead to unnecessarily long response times and in turn will increase computation time.

In a conventional central-cloud system, all the data processing at a central place causes huge data transmission traffic, which also leads to unnecessarily long response times and in turn, will increase computation time. It is forecasted that in [8]–[11], for the 2025 time frame, the number of connected vehicles will grow to about 100 million globally and the data volume delivered between vehicles and the cloud will be about 100 petabytes per month. Assuming 20GB per month per vehicle and three million vehicles (12% market share and 25% regional ratio of 100 million vehicles), 60 petabytes of vehicle data will come to the cloud every month. Assuming the data transaction rate at the cloud is 10GB per second, it will take 70 days just for the transactions. For this reason, to be able to establish a practical platform to serve Vehicle-to-Cloud (V2Cloud) services, both computation and network performance need to be taken into account.

Edge computing is expected to reduce the network latency and mobile resource demands by migrating the computing and storage capabilities from the central cloud to the edge of a mobile network [12]. It has been one of the key emerging technologies. In an edge computing framework, computation power can be deployed as close as possible to the vehicles, achieving better performance for these latency-sensitive and location-aware applications.

How to deploy edge computing is still an open issue. The most common solution is to deploy edge computing along with the base station. However, it will increase the complexity of the base station. Another possible solution is CUPS, i.e. Control and User Plane Separation, which is supposed to be one of the key architectures for future cellular networks. It enables the core network functionality separation, the network can be deployed and operated flexibly without affecting the functionality of the existing nodes subject to this split.

As shown in Figure 1, CUPS allows operators to separate the EPC (evolved packet core), typically the SGW (serving gateway), PGW (packet data network gateway), into the control plane and user plane. The control plane can sit in a centralized location, for example, the middle of the country, and the user plane can be placed in proximity to the application it is supporting. Despite this separation, the functionality of the existing nodes subject to this split is not influenced. As a result, CUPS supports the increase of data traffic by enabling to add more user plane nodes without changing the number of control plane nodes and reduces latency on application service by placing gateways as close as possible from the radio access networks or users. Therefore, CUPS is one of the indispensable technologies to enable edge computing and achieve the QoE (quality of experience) anticipated with the 5G introduction.

In detail, CUPS allows for reducing latency on application service, for example, we can place user plane nodes in proximity to the applications they are supporting. And by enabling adding user plane nodes, CUPS supports increased data traffic. Besides, by CUPS, we can scale the control plane and user plane resources of the EPC nodes independently and enable software-defined networking (SDN) to deliver user plane data more efficiently. By flexibly deploying user plane nodes, we can offload the information processing at the local edge server instead of concentrating all the information at the central server. This mitigates the burden of both networks and the central cloud. Besides, we can shorten the response latency because: (i) the user plane nodes are isolated from each other and, (ii) the edge server is deployed at a much closer place than the central server's location, making it possible to collect and distribute information from a local edge server in a more timely manner.

III. DESIGN

In this section, we introduce the design of deep learning-based application-specific control and user plane separation (CUPS) edge computing architecture.

As shown in Figure 2, by applying control and user plane separation, the Evolved Packet Core (EPC) control plane could be virtualized and located in a cloud environment while

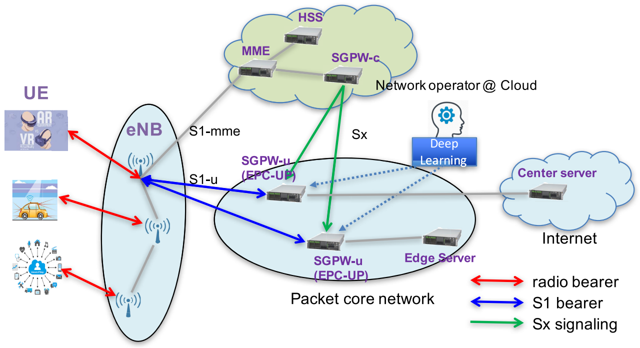


FIGURE 2. Deep learning-based CUPS architecture for edge computing [13].

the user plane can remain in the transport network. We deploy multiple user planes on-demand for a single control plane. For each UE, we set up one radio bearer and multiple S1 bearers for multiple EPC user planes. Compared to the multiple-bearer network architecture defined in 3GPP [14], where UE needs to monitor and select an optimal radio bearer for its packet based on the QoS classes, the merit of our single radio bearer architecture can eliminate the workload of UE side and save its battery life. Another merit of design is that we can support multiple applications while the conventional multiple bearer based architecture [14] can only support a maximum of 11 classes of traffic.

Next, how could the eNB classify the packets according to the application and select the proper S1 bearer for the radio packet is challenging here due to two main challenges. First, as long as data has been transmitted from user equipment (UEs) into a mobile network, the contextual information of the data (e.g., which application the data belongs to) is hidden from network alliances. Second, data packets will be compressed, concatenated, and modulated in a RAN area, which makes application identification in a RAN much more difficult than that in a core network (CN).

In our previous work [15], [16], we have proposed the deep learning-based application identification architecture, where a small number of customized supervising phones are used to generate training data in real-time and apply deep learning at the packet gateway (P-GW). We reuse the traffic classification architecture to tag the downlink packets with `app_name` from P-GW to eNB. There are two benefits of the design: (1) We don't need to apply deep learning-based application identification on eNB so that the overload of eNB could be reduced; (2) The traffic of all UEs must pass through the P-GWs while there is only a very limited number of UEs connected a single eNB so that we may not be able to get enough training data if we apply machine learning on the traffic collected at a single eNB.

Machine learning-based identification has been proposed in [17]. The selected features are fed into some kind of classifiers such as Naive Bayes [18], K-Means [19], and Neural Network [20]. Conventional machine learning techniques are

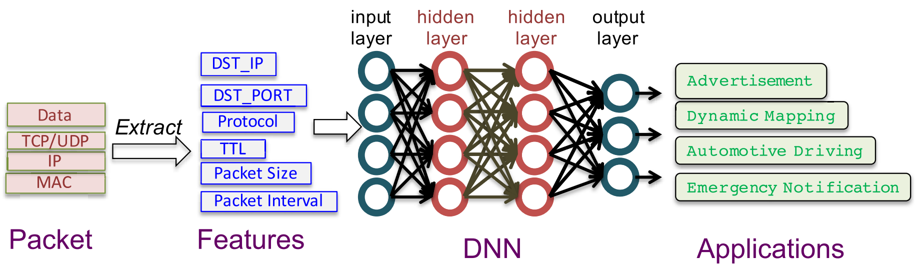


FIGURE 3. Application identification with deep neural networks with extracted features.

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 is proposed to replace handcrafted features with efficient algorithms for feature learning and hierarchical feature extraction.

As shown in Fig. 3, our training model is defined based on deep neural networks (DNN) with an input layer, multiple fully connected hidden layers, and an output layer. Each hidden layer is a feed-forward neural network.

We extract feature vectors from packets and feed the vectorized features $\mathtt{X}$ into the input layer. Each hidden layer takes the previous layer output $\hat{\mathtt{Y}}$ as input $\mathtt{X}$ and multiplying it by weight matrices $\mathtt{W}$, add bias vector $\mathtt{b}$ associated to those inputs ($\hat{\mathtt{Y}} = \mathtt{W} \cdot \mathtt{X} + \mathtt{b}$). The output layer computes the softmax probabilities that are assigned to each application. We use the cross-entropy loss function as a cost function and use stochastic gradient descent as an optimization algorithm to minimize the cost during training.

Comparing to conventional work on identifying applications from the traffic trace relies on DPI of the user data, our application identification 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.

A. SELECTION OF FEATURE VECTORS

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. Besides the above three features, `TTL` is 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. Similarly, the packet interval between two sequential packets of both uplink and downlink packets of a flow. We also find that packet size is a useful feature. This is because clients and servers need to exchange information during connection establishment. The size of the exchange information is usually application-specific. Therefore, we add sizes of the first few packets of each flow as

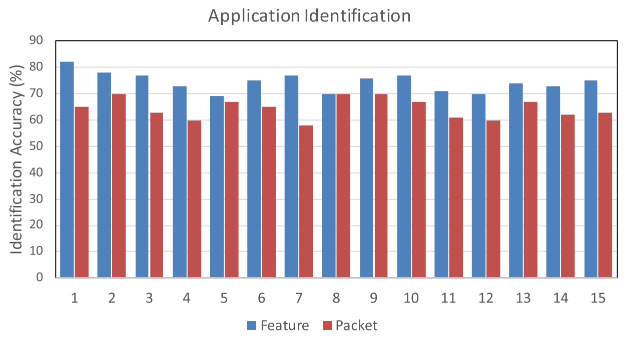


FIGURE 4. Experimental results of application identification over mobile traffic from the OPTAGE MVNO.

a useful feature to our training model. In summary, we use a vector of `server_ip`, `server_port`, `proto`, `TTL`, `packet_size`, and `packet_interval` as features to identify applications. We mark it as feature-based. As a comparison, without feature extraction, we use the first 60-bytes of the uplink and downlink packets as the input feature vector directly, we mark the method as payload-based.

B. EXPERIMENTS OF APPLICATION IDENTIFICATION

We begin with a brief introduction of our GPU DNN platform, consisting of one Intel 8-core Xeon E5-2670 2.60GHz processor and one NVIDIA GTX1080 card, 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 a 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 using CPU to preprocess and slice training data into GPUs and coordinate training parameters during training. We use two-week (1-14 September 2019) OPTAGE [21] MVNO data as training data, where each day of traffic consists of about 20000 flows. Besides training and inference, we use one-day (15 September 2019) traffic as validation in training.

As shown in Figure 4, by using a tuple of < `dst_ip`, `dst_port`, `proto`, `ttl`, `packet_size`, `packet_interval` > as the features of application traffic captured at an MVNO, we can successfully identify 200 mobile applications with about 75% accuracy over 15-day (16-30 September 2019) traffic using an 8-layer Deep Neural Network with TensorFlow [22]. As a comparison, if we use payload as an input vector without feature extraction, we can only achieve an identification accuracy of about 65%. The experimental results show that feature extraction is important in deep learning-based application identification due to the fact that less misleading data can improve the accuracy.

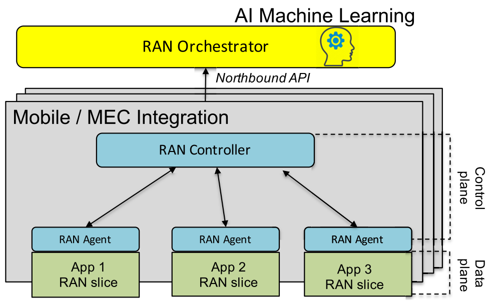


FIGURE 5. Architecture of application-specific RAN slicing.

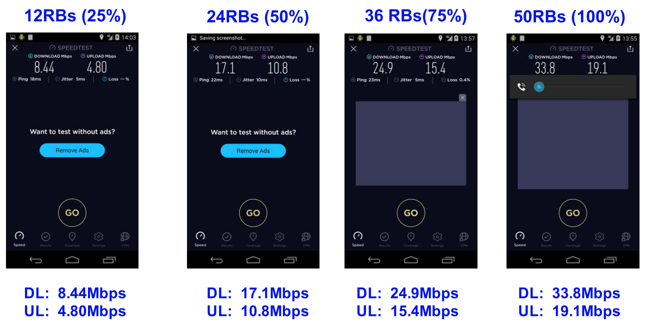


FIGURE 6. Throughput of RAN slices with different assigned RBs.

IV. IMPLEMENTATION

In this section, we introduce the detailed implementation of classifying and diverting packets to application-specific edge servers from the RAN to the core network via network slicing. Network slicing has been considered as one of the most significant technologies for 5G mobile networks [3], where multiple slices that support different categories of services with different quality of service (QoS) requirements are supposed to be deployed in the same physical infrastructure.

A. APPLICATION-SPECIFIC RAN SLICING

RAN (Radio Access Network) Slicing is a mandatory component of the end-to-end networking and recently catches much attention both in academia and in industries. Radio resources represented as Resource Blocks (RBs) may be isolated and allocated to UEs applications, and services to enable resource isolation for achieving desired QoS in RAN. While the basic concept of RAN slicing has been already addressed in our various research projects such as 5G! Pagoda and industry collaborations, there are more and more interesting research challenges to be addressed.

For different applications, in this subsection, we propose that the packets should not only be diverted to different EPC user planes but also be able to be assigned to different RAN slices with different radio resource blocks (RBs) and spectrum scheduling algorithms. For example, autonomous driving requires low latency while video streaming requires high-bandwidth.

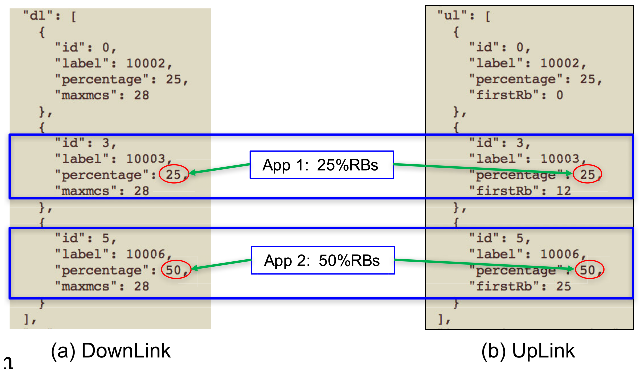
FlexRAN [23] to our knowledge is the most complete implementation of RAN slicing where the RAN control and data planes are decoupled through a custom southbound API. Each RAN slice could have its own spectrum resource blocks (RBs) and MAC scheduling algorithm, which could be configured remotely in real-time through the FlexRAN agents, which are composed of Virtual Subsystem Functions (VSFs) responsible for control operations such as RAN RBs scheduling. The controller can swap VSFs dynamically and program RAN slices via the config file at runtime.

However, in naive FlexRAN, we can only do UE-specific RAN slicing where we assign UEs to each RAN slice at a granularity of UE, which lacks flexibility. As shown in Figure 5, we implement application-specific RAN slicing [24] as an extension to a modified version of the FlexRAN, where we reuse the deep learning-based traffic classification architecture to tag the downlink packets with `app_name` from P-GW to RAN. The workflow of application-specific RAN slicing is as follows:

- For each application-specific RAN slice with `slice_id`, we assign it with a `label` in the `ran-slicing-config.json` file. The `label` is calculated as a hash of the `app_name`.
- When receiving a packet from P-GW, eNB checks the tagged `app_name`, calculates `label` by hashing `app_name`, and then looks up the table <`label`, `slice_id`> to get `slice_id`.
- The eNB updates the mapping of UE and `slice_id` at the interval of every radio frame (10ms in LTE).

In Figure 6, we examine whether we can assign packets from a mobile application (e.g., `App1`) to a specific RAN slice and whether we can modify the assigned RBs of the RAN slice in real-time. We define two slices: default and `App1` slices. For both downlink and uplink, we assign a different number of RBs to the `App1` slice and check the throughput of `App1`. The experimental result shows that the throughput of both uplink and downlink is proportional to the assigned RBs.

Figure 7 shows an example of ran-slicing-config.json file. Here, we define three slices with *slice_id* 0, 3, and 5, where slice 0 is defined as the default slice with 25% RBs, slice 3 as label 10003 with 25% RBs and slice 3 is defined as label 10006 with 50% RBs. We assign packets of application `App 1` to slice 3, `App 2` to slice 3, and all other packets to slice 0.



```
"dl": [
  {
    "id": 0,
    "label": 10002,
    "percentage": 25,
    "maxmcs": 28
  },
  {
    "id": 3,
    "label": 10003,
    "percentage": 25,
    "maxmcs": 28
  },
  {
    "id": 5,
    "label": 10006,
    "percentage": 50,
    "maxmcs": 28
  }
],
```

```
"ul": [
  {
    "id": 0,
    "label": 10002,
    "percentage": 25,
    "firstRb": 0
  },
  {
    "id": 3,
    "label": 10003,
    "percentage": 25,
    "firstRb": 12
  },
  {
    "id": 5,
    "label": 10006,
    "percentage": 50,
    "firstRb": 25
  }
],
```

FIGURE 7. Example of `ran-slicing-config.json` with three application-specific RAN slices.

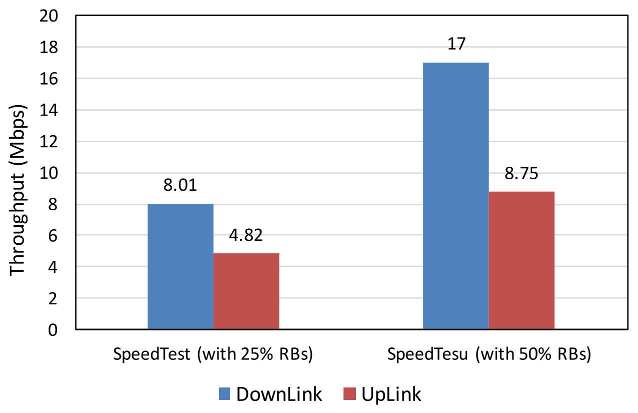


FIGURE 8. Throughput of `SpeedTest` and `SpeedTesu` slices with 25% and 50% RBs separately.

To compare the applications in different RAN slices fairly, we use `AppCloner` to clone `SpeedTest` to a new application named `SpeedTesu`, which is identical to `SpeedTest` except the application name. According to our test, `SpeedTest` and `SpeedTesu` achieve the same throughput when they are in the same RAN slice. We define three slices: default, `SpeedTest` and `SpeedTesu` slices. As shown in Figure 8, we assign 25% and 50% of RBs to `SpeedTest` and `SpeedTesu` slices separately. The experimental results show that `SpeedTesu` slice can get twice the throughput of that of `SpeedTesu` slice due to that it has twice the RBs.

B. APPLICATION-SPECIFIC EPC SLICING WITH CUPS

In this subsection, we introduce our research efforts on prototyping Application-Specific CUPS using deeply programmable network nodes with general-purpose processors and network processors, called FLARE programmable nodes [25] and FPGA boards with OpenAirInterface (OAI) [26] on top of them.

OpenAirInterface (OAI) is an open experimentation and prototyping platform created by EURECOM. It provides a software implementation of all elements of the 4G LTE/5G architecture including user equipment (UE), eNodeB

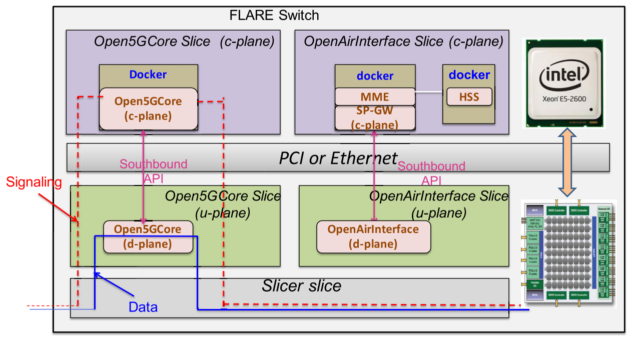


FIGURE 9. Architecture of EPC slicing supporting CUPS [3].

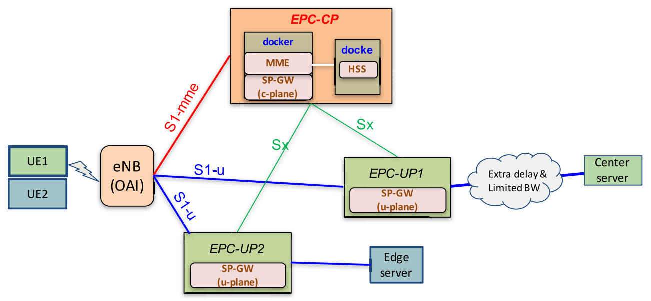


FIGURE 10. CUPS architecture with multiple user planes.

(eNB), Home Subscriber Server (HSS), and Evolved Packet Core (EPC) components. A compound EPC component consists of Serving Gateway (S-GW), Packet Data Network Gateway (P-GW) as well as the Mobility Management Entity (MME). The eNB and EPC components are responsible for creating channels (namely bearers) with UE and forwarding the user traffic.

As shown in Figure 11, we implement a software EPC on FLARE [3], where signaling related EPC entities (e.g., MME, HSS, SP-GW-c) are implemented in control plane while user data forwarding and processing (e.g., SP-GW-u) are implemented in the data plane. The data plane and user plane connected via the PCI interface.

In this paper, we extend the work [3] by running the EPC control plane and EPC user plane on different machines, where the data plane and control plane are connected via the Internet. As shown in Figure 10, we run the signaling entities of an EPC slice, e.g., MME and the control plane of SP-GW, in a Docker instance. We can also run the HSS entity in another Docker instance. These two docker instances are isolated and replaceable without interfering with each other. For example, we can install different versions of packages in MME and HSS instances even if they may conflict with each other when installed on the same host machine.

We implement SP-GW (u-plane) components with chained Click elements on many-core processors for high-performance, which is implemented with GTPV1-U kernel module in naive OAI software. To scale network functions, we subdivide and modularize the EPC user plane into network

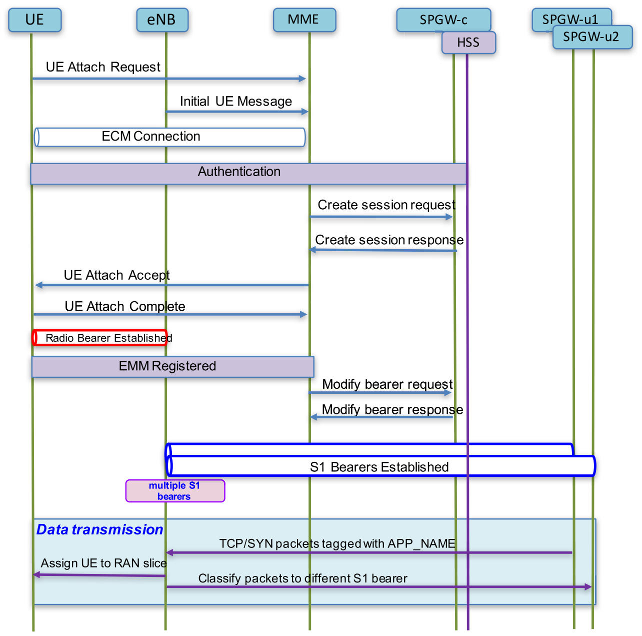


FIGURE 11. Signaling flow for bearer establishment in designed CUPS.

functions and to parallelize packet processing across on-chip multiple processors.

The Sx signaling is defined following OpenFlow's convention for programming abstraction. We define our own programming abstraction as API as follows,

<**UEID, TEID**><**Action**><**Stat**>,

where **UEID** may be a UE's IP address assigned by MME through the signaling channel and **Action** may be actions such as create/ update/ remove a *GTP-U* tunnel.

The 3GPP defines a new Packet Forwarding Control Plane (PFCP) protocol for communication between the control plane and the user plane. According to 3GPP TS 29.244 [14], PFCP is mainly responsible for:

- Create an association between UP function and CP function over UDP port 8805;
- Enable CP function to control UP function on how to process certain traffic;
- Forward packets between CP and UP;
- Select UP function based on DNS.

Note that the signaling flow of our prototyped CUPS is different from 3GPP-PFCP. The protocol between CP and UP is implemented over TCP instead of UDP in 3GPP-PFCP, where we don't need to define extra association and keep-alive schemes between CP and UP. All other basic functions of PFCP have been implemented in our prototype.

As shown in Figure 11, signaling flow for bearer establishment in our designed CUPS is as follows:

When a UE sends a UE_ATTACH_REQUEST to eNB, eNB will authenticate the UE and encapsulate the UE_ATTACH_REQUEST into a CREATE_ SESSION_ REQUEST and sends it to the EPC-CP. The EPC-CP can parse the UE_ATTACH_ REQUEST and get the embedded APN, IMSI, and eNB info. EPC-CP can select EPC-UP based on one or combination of this info. In our implementation, we use APN info as an example. Unlike 3GPP-PFCP, we don't need any DNS procedures here. Instead, we use the keywords embedded in the APN info. For example, if the APN name is "vehicle.map", we can select all user planes related to the vehicle mapping applications. Of course, the EPC-CP needs to maintain a database about the mapping of the <APN, UP(s)>.

After the UE receives the UE_ATTACH_ACCEPT message from eNB, it will send back a UE_ATTACH_COMPLETE message and establish a radio bearer between the UE and the eNB.

After EPC-CP receives MODIFY_BEARER_REQUEST, it will send GTPU configuration to all selected EPC user planes via the customized southbound API. The format is < UEID,TEID><Action>, where UEID may be a UE's IP address assigned by MME through the signaling channel and Action may be actions such as create/update/ remove a GTP-U tunnel. Then one S1 bearer is established between eNB and each EPC user plane. The eNB will get the selected EPC-UP info after it receives the GTP-U packet from an S1 bearer.

When a packet from the UE arrives at the eNB, we identify the application name of the packet at the eNB and classify it to the corresponding S1 bearer and send it to the corresponding EPC user plane.

C. APPLICATION-SPECIFIC EDGE SERVER SLICING

Even for one physical edge router, we can also apply application-specific slicing to slice an edge server into multiple MEC slices so that each MEC slice can serve one application separately. As shown in Figure 12 [12], we use two FLARE nodes: one is to classify traffic from RAN to different virtual network functions NFV_VLAN while the other is to classify reverse traffic accordingly. The traffic of NFV_VLANs is isolated using VLANs on the hosting MEC server running Open vSwitch [27]. Packets from smartphones are classified and tagged with different VLAN IDs according to applications at the FLARE2 and then are diverted to the MEC server. In each NFV_VLAN, we apply a 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).

The application-specific MEC slicing is useful for application developers, as unlike conventional LTE networks where only operators can define what capabilities for MEC network functions to be exposed to third parties, application developers may add arbitrary functionalities, especially of ultra reliable and low latency communication (uRLLC).

D. EDGE SERVER DEPLOYMENT

In mobile edge computing, we can deploy a set of edge servers geographically. The mobile users can access the nearest edge servers with the lowest latency. One fundamental

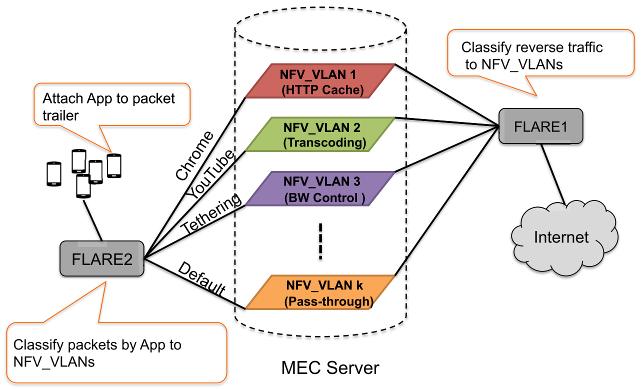


FIGURE 12. Architecture of application-specific edge server slicing [12].

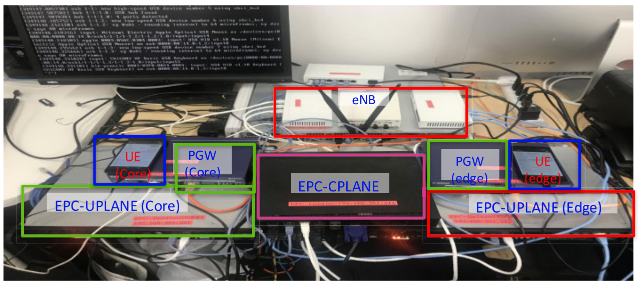


FIGURE 13. Prototype of CUPS based edge computing [13].

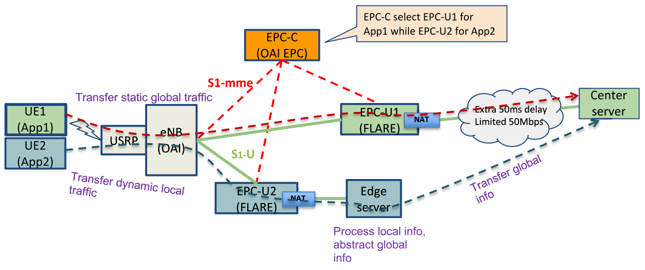

EPC-C select EPC-U1 for App1 while EPC-U2 for App2
EPC-C (OAI EPC)
S1-mme
Transfer static global traffic
UE1 (App1)
UE2 (App2)
USRP
eNB (OAI)
S1-U
EPC-U1 (FLARE)
NAT
Extra 50ms delay Limited 50Mbps
Center server
Transfer dynamic local traffic
EPC-U2 (FLARE)
NAT
Edge server
Transfer global info
Process local info, abstract global info

FIGURE 14. Experimental setup with offloading traffic to edge server.

problem is how to select edge servers for many mobile users so that the total waiting time is minimized.

The control plane is responsible for selecting a P-GW according to a UE's request. Whenever a UE issues a request for a service, the control plane will select the optimal edge server in terms of service time.

We assume that the system consists of N geographically distributed edge servers, where each edge server i has a processing capacity Ci. The transmission delay between the UE i and edge server j is D_{ij}, the queueing delay at server j is Q_j while the processing delay at server j is P_{ij}. So the control plane will select the edge server with the minimum delay $T_i = \arg\min(D_{ij} + Q_{ij} + P_{ij})$ for UE i.

Since the location of each edge server j is fixed, the control plane can calculate the D_{ij} based on the location of UE i. So we only need to consider the queueing delay Q_{ij} and processing delay P_{ij} at server j.

We assume that requests from the UE follow a Poisson process with rate λ, which represents the average number of requests per unit time. We assume the service time of the request (e.g., the flow size of dynamic mapping information) follows an exponential distribution $1/\mu$, where μ is the service rate of the edge server. We assume the edge server can serve c requests simultaneously. We define $\rho = \lambda/c\mu$, $\rho < 1$. Otherwise, the edge server will be overloaded. Then the edge server can be modeled as a traditional Erlang's C formula.

According to the above analysis, we have the following conclusion.

- Where there are less than c quests being processed in all edge servers, the queueing delay Q_{ij} is equal to 0. P_{ij} is equal to the service time of the request. In this case, the control can simply select the edge server with the smallest transmission latency. E.g., edge server close to a mobile vehicle.
- Where there are more than c quests being processed at the nearest edge servers, we need to consider the queueing delay Q_{ij}. Since the processing delay P_{ij} only depends on the requests itself, which is independent of the server. The control plane needs to choose an edge server with the $\min(D_{ij} + Q_{ij})$. The D_{ij} can be evaluated by the distance between the UE i and edge server j, while the queueing delay Q_{ij} can be evaluated by the Erlang Formula $\rho/\lambda \cdot \rho/(1-\rho)$.

V. EVALUATION

In this section, we evaluate our prototyped application-specific CUPS system shown in Figure 13. Since we have evaluated the performance of application-specific RAN slicing in Section IV-A, we focus on evaluating the scenarios of application-specific data processing without or with CUPS.

A. LOCAL CUPS TESTBED IN LAB ENVIRONMENT

We use USRP B201 as a software radio platform and two Nexus5 as UEs. The EPC control plane is a modified OpenAirInterface EPC while two EPC user planes are implemented on FLARE [13], [28]. We use two local servers to emulate the edge server and the central server. Comparing to the edge server, we assume the central server is suffering from extra delay and bandwidth limitation due to the fact that a lot of UEs may collect and send traffic to the central server simultaneously. In our evaluation, we set the extra delay to 50ms while the limited bandwidth to 5Mbps.

We assume `UE1 (App1)` is for static information that should be stored in the central server while `UE2 (App2)` is for some local dynamic information that could be offloaded to the edge server for processing. We evaluate two scenarios as shown in Figure 14: 1) without edge server to offload traffic, where all traffic from `UE1 (App1)` and `UE2 (App2)` is sent to the central server; 2) with edge server to offload traffic, where traffic to `UE2 (App2)` will be sent to the edge server first and then only abstracted info will be sent to the central

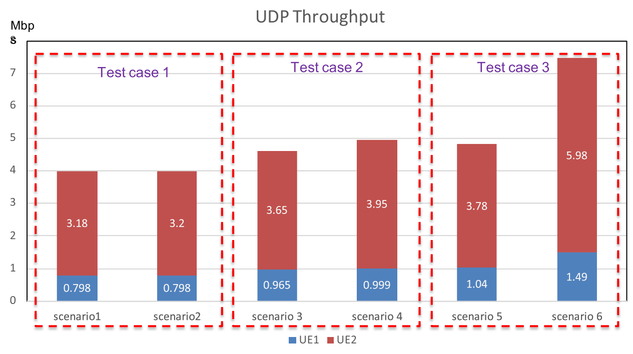


FIGURE 15. Evaluation of data processing with and without edge computing.

server, where the abstracted info is much smaller comparing to original info.

We emulate the data collection procedure from UEs to the edge and the core server.

Test Case 1 (traffic volume is small): we set the total send rate as 4Mbps (UE1: 0.8Mbps, UE2: 3.2Mbps). The total traffic volume is smaller than the capability of the input link of the central server. We can observe the packet loss ratio is very small and there is almost no difference between the total traffic volume received at the central server in both scenarios.

Test Case 2 (traffic volume is medium): we set the total send rate as 5Mbps (UE1: 1Mpbs, UE2: 4Mbps). The total traffic volume is almost equal to the capability of the input link of the central server. We can observe a small number of packet losses (15%) in scenario 1, while there is almost no packet loss in the scenario.

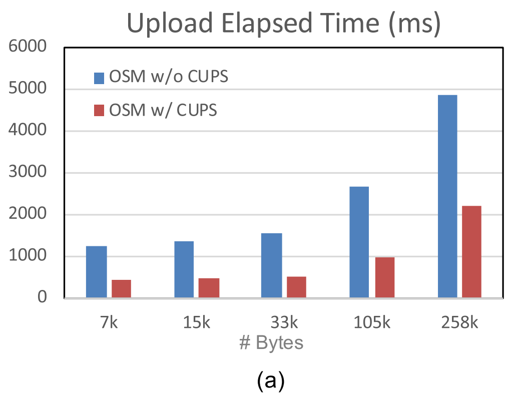
Test Case 3 (traffic volume is heavy): we set the total send rate as 7.5Mbps (UE1: 1.5Mbps, UE2: 6Mbps). The total traffic volume is much higher than the capability of the input link of the central server. We can observe a large number of packet losses (45%) in scenario 1, while there is almost no packet loss in scenario 2. The experimental results show that our implemented CUPS based edge computing system achieves a significant decrease in transmission data to the central server so that we can handle much more sensor data.

The experimental results show that our implemented CUPS based edge computing system achieves a significant decrease in transmission data to the central server so that we can handle much more sensor data.

B. USE CASE: DYNAMIC MAPPING

Next, we use dynamic mapping as an example of use case to evaluate the above prototyped CUPS testbed.

Dynamic mapping plays an indispensable role in future intelligent transportation systems and autonomous vehicles. It is a database that consolidates static information such as 3D structures and dynamic information, for example, nearby vehicles, traffic signals, and road works. With dynamic mapping, a vehicle would know its relevant road information in advance. In general, data is collected from onboard measuring



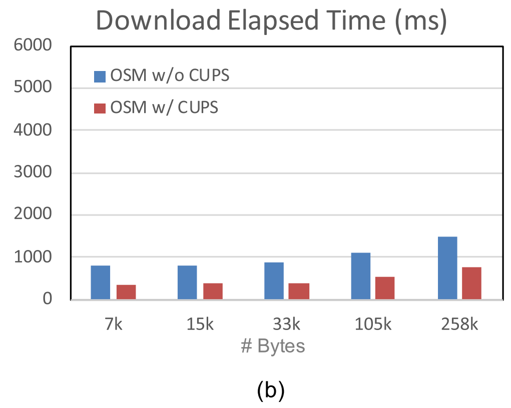


FIGURE 16. Evaluation of (a) Upload and (b) Download time of dynamic mapping information versus the flow size of transmitted information.

instruments like cameras, radar sensors, and laser scanners (LIDAR), transferred and processed in the cloud. Different from traditional maps, the dynamic map must be precise enough to accurately localize dynamic objects, so a large amount of data transfer is necessary to update the map. Besides, as a result of the dynamic information that needs to be updated on time, low latency is especially required. In this sense, the edge computing system with CUPS-based data offloading is a very promising technology to fulfill above strict requirements for dynamic mapping.

We install OpenStreetMap [29] server software on both central and edge servers. We use a smartphone to take photos of moving cars and traffic lights as the dynamic mapping information and upload them to central and edge servers to show on a static OpenStreeMap there.

We use UE to upload and download pictures (dynamic mapping information) with different sizes to and from OSM servers. Then we measure the elapsed time. The result and comparisons of with and without CUPS are shown in Figure 16. The experimental results in Figure 16 show the elapsed upload and download time of data from users versus the size of information.

The experimental results in Figure 16 shows the elapsed time of data uploaded from users versus the number of

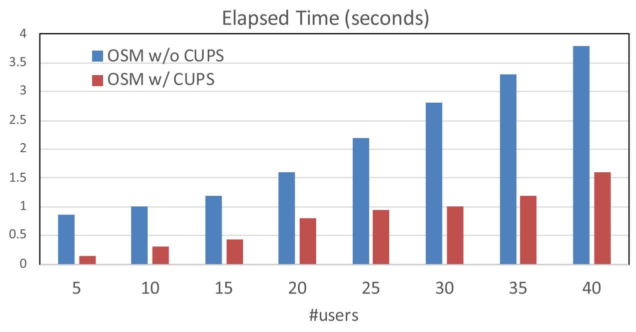


FIGURE 17. Evaluation of upload time of dynamic mapping information versus the number of users [13].

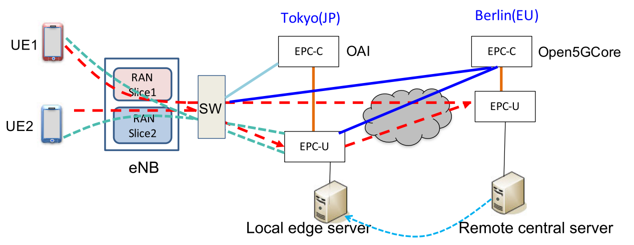


FIGURE 18. Architecture of international CUPS testbed.

simulated users (vehicles or other data collection devices). Compared with transferring raw data collected by vehicles directly to the central server (conventional approach), this CUPS-based system saves much upload time, thereby supports the increase of data transfer, and reduces the burden to the core network.

C. INTERNATIONAL CUPS TESTBED BETWEEN EU AND JP

In the previous subsection, we have verified our proposed CUPS architecture in a lab environment, where we add an extra delay to emulate the latency difference between edge and cloud. In this subsection, we evaluate the CUPS architecture in our international testbed. As shown in Figure 18, we set up two user planes: one is in Tokyo (Japan), another is in Berlin (Europe).

It is very important to efficiently use the network resources when considering the huge demand for data transfer for the connected vehicles. Future connected and autonomous vehicles will generate various kinds of data such as CAN data, camera data. Those data usually have diverse QoS requirements depending on the application provided. Application-specific CUPS is the key for efficient use of mobile network resources, not only offloading data to alleviate network congestion, but also intelligently offloading according to different applications to better satisfy their application requirements.

In Figure 19, we compare the RTT and TCP throughput between two UEs when without and with CUPS. The experimental results show that RTT between UEs can be reduced

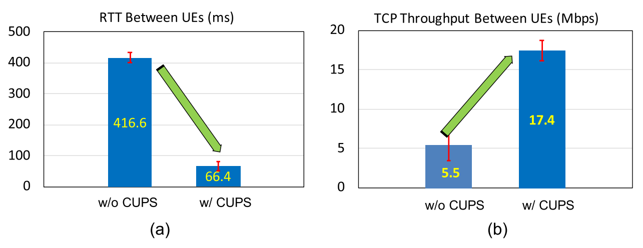


FIGURE 19. Evaluation of international CUPS testbed in terms of (a) RTT and (b) TCP throughput.

from 416.6ms to 66.4ms while the TCP throughput can be increased from 5.5Mbps to 17.4Mbps.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose an application-specific edge computing network architecture with CUPS in mobile core networks. We apply deep learning to classify packets from different applications to different Radio Access Network (RAN) slices for application-specific spectrum scheduling and also route packets to different edge servers for application-specific processing. The capabilities of our system have been evaluated and the obtained experiments results have shown that our system achieves a significant decrease in transmission data and latency. Our future work will focus on implementing more real use cases.

REFERENCES

[1] 5GMF White Paper. (Jul. 2016). *5G Mobile Communictions Systems for 2020 and Beyond*. [Online]. Available: http://5gmf.jp/wp/wp-content/uploads/2016/09/5GMF_WP101_All.pdf

[2] *The 5G Infrastructure Public Private Partnership*. Accessed: Sep. 15, 2021. [Online]. Available: https://5g-ppp.eu/

[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," *J. Inf. Process.*, vol. 25, pp. 153–163, Mar. 2017.

[4] A. Callado, C. Kamienski, G. Szabo, 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, 3rd Quart., 2009.

[5] 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," Tech. Rep. ietf rfc6335, 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6335

[6] T. Erpek, A. Abdelhadi, and T. C. Clancy, "An optimal application-aware resource block scheduling in LTE," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2015, pp. 275–279.

[7] J. He and W. Song, "AppRAN: Application-oriented radio access network sharing in mobile networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 3788–3794.

[8] Cisco. (2011). *A Business Case For Connecting Vehicles*. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/mfg/Connected-Vehicles_Exec_Summary.pdf

[9] GSMA. (2012). *2025 Every Car Connected*. [Online]. Available: https://www.gsma.com/iot/wp-content/uploads/2012/03/gsma2025everycarconnected.pdf

[10] SBD. (2015). *Connected Car Global Forecast 2015*. [Online]. Available: https://www.sbdautomotive.com/

[11] PWC. (2016). *Connected Car Report 2016*. [Online]. Available: https://www.pwc.nl/en/publicaties/connected-car-report-2016.html

[12] P. Du and A. Nakao, "Application specific mobile edge computing through network softwarization," in *Proc. 5th IEEE Int. Conf. Cloud Netw. (Cloudnet)*, Oct. 2016, pp. 130–135.

[13] P. Du, A. Nakao, Z. Sun, L. Zhong, and R. Onishi, "Deep learning-based C/U plane separation architecture for automotive edge computing," in *Proc. 4th ACM/IEEE Symp. Edge Comput.*, Nov. 2019, pp. 295–297.

[14] (2017). *Interface Between the Control Plane and the User Plane of EPC Nodes*. [Online]. Available: https://www.3gpp.org/DynaReport/29244.htm

[15] A. Nakao and P. Du, "Toward in-network deep machine learning for identifying mobile applications and enabling application specific network slicing," *IEICE Trans. Commun.*, vol. 101, no. 14, pp. 153–163, 2018.

[16] P. Du and A. Nakao, "Deep learning-based application specific RAN slicing for mobile networks," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2018, pp. 1–3.

[17] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Proc. IEEE Conf. Local Comput. Netw. 30th Anniversary (LCN)*, Nov. 2005, pp. 250–257.

[18] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: When randomness plays with you," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 37–48, 2007.

[19] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core," in *Proc. 16th Int. Conf. World Wide Web (WWW)*, 2007, pp. 883–892.

[20] 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, Jan. 2007.

[21] *Optage Inc*. Accessed: Sep. 15, 2021. [Online]. Available: https://optage.co.jp/en/

[22] *Tensorflow*. Accessed: Sep. 15, 2021. [Online]. Available: https://www.tensorflow.org/

[23] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *Proc. 12th Int. Conf. Emerg. Netw. Exp. Technol.*, 2016, pp. 427–441.

[24] P. Du and A. Nakao, "Understanding intelligent RAN slicing for future mobile networks through field test," in *Proc. 20th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2019, pp. 1–6.

[25] A. Nakao. (2012). *Flare: Open Deeply Programmable Network Node Architecture*. [Online]. Available: http://netseminar.stanford.edu/seminars/10_18_12.pdf

[26] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 33–38, 2014.

[27] *Open Vswitch*. Accessed: Sep. 15, 2021. [Online]. Available: http://openvswitch.org/

[28] Z. Sun, P. Du, A. Nakao, L. Zhong, and R. Onishi, "Building dynamic mapping with CUPS for next generation automotive edge computing," in *Proc. IEEE 8th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2019, pp. 1–6.

[29] OpenStreetMap. (2007). *Openstreetmap*. [Online]. Available: https://www.openstreetmap.org/

PING DU (Member, IEEE) received the Ph.D. degree from the Graduate University for Advanced Studies, Japan, in 2007. Since 2008, he has been working with the National Institute of Information and Communication Technologies (NICT), Japan. He currently works with The University of Tokyo as a Project Associate Professor. His research interests include optical networks, network security, network virtualization, mobile networks, and machine learning.

AKIHIRO NAKAO (Member, IEEE) received the B.S. degree in physics and the M.E. degree in information engineering from The University of Tokyo, and the M.S. and Ph.D. degrees in computer science from Princeton University. He worked at IBM Yamato Laboratory, Tokyo Research Laboratory, and IBM, Austin, TX, USA. Since 2005, he has been an Associate Professor and then a Professor in applied computer science with the Interfaculty Initiative in Information (III) Studies, Graduate School of Interdisciplinary Information Studies, The University of Tokyo. He is currently a Professor with the Faculty of Engineering, The University of Tokyo. He has been appointed as the Chairman of the 5G Mobile Network Promotion Forum (5GMF) Network Architecture Committee by the Government of Japan.

LEI ZHONG received the Ph.D. degree in informatics from the Graduate University for Advanced Studies, Japan.

From 2011 to 2017, he was involved in many national research projects related with wireless communication and networking when he worked with the National Institute of Information and Communications Technology (NICT), the National Institute of Informatics (NII), and The University of Tokyo. He is currently a Senior Researcher with Toyota Motor Corporation, where he is responsible for the research and standardization of future network and computing infrastructure for connected vehicles. He is also serving as a Toyota Delegate in several related standardization organizations, such as 3GPP and AECC. His research interests include edge computing, vehicular networking, software-defined networking, the Internet of Things, and big data. He has published more than 30 technical papers and applied several patents in these research fields.

RYOKICHI ONISHI received the B.S. and M.S. degrees in information and communication engineering and the Ph.D. degree in electrical engineering and information systems from The University of Tokyo. Since 2001, he has been working at Toyota InfoTechnology Center and Toyota Motor Corporation, where he is currently the Manager of End-to-End Computing Group, Connected Advanced Development Division. His current interests include the IoT and mobile communication technologies geared for emerging vehicle services and has been awarded 30 patents by the U.S. Patent Office in this domain.