Application-Specific Mobile Network through In-Network Deep Learning

Ping Du

The University of Tokyo

22 June 2018@BUPT
P-GW is the best point to perform application identification since all traffic needs to go through it.

P-GW can convey its identified app-info to both RAN and CN.
1. In-Network Deep Learning based Application Identification
Application Identification with DNN

Feature Extraction

Flow Features

Application Identification

0.05 (App 1)
0.05 (App 2)
0.80 (App 3)
0.02 (App 4)
0.03 (App 5)
0.05 (App 6)
Intuitive Rationale for Deep Neural Networks
( Multi-layer Spatial Transformation)
Features of Applications

**Human**

- eyes
- ears
- mouth
- nose
- shape

**DNA == PAYLOAD**

**Applications**

- dst_ip
- dst_port
- ttl
- protocol
- packet size

**Packets**
Parallelly training multiple DNNs
Features Extraction with DNN (1)

Feature Vector 1 = [server_ip, server_port, proto]
Feature Vector 2 = [client_ip, client_port, server_ip, server_port, proto]

Conclusion:

unimportant features: client_ip, client_port
Features Extraction with DNN (2)

Feature Vector 1 = [server_ip, server_port, proto]

Impact of TTL

<table>
<thead>
<tr>
<th>Vector 1</th>
<th>Vector 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.61</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Impact of packet size

<table>
<thead>
<tr>
<th>Vector 3</th>
<th>Vector 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.81</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Vector 3 = [Vector 1, TTL]

Vector 4 = [Vector 3, packet_size]

Conclusions:
Important features: ttl, packet_size
Preliminary experimental results

Accuracy of App Identification over 39-day traffic [avg. 93.5%]
2. Application-Specific Data Processing
Application-Specific MEC Processing

Classify packets by App to NFV_VLANs

Attach App to packet trailer

Classify reverse traffic to NFV_VLANs
User Interfaces of App-specific MEC

1. Web-GUI based App control
   - Web-GUI based App control

2. Command line based App control
   - $ appctrl set 'com.android.chrome output 1'
   - $ appctrl del 'com.android.chrome'
   - $ appctrl dumprules

(SDN)  <Flow><Action><Stat>

(MEC)  <App><NFV><Stat>
Application-Specific QoS Control

NakaoLab UTokyo MVNO

List of current rules:

<table>
<thead>
<tr>
<th>App name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.facebook.katana</td>
<td>Pass through</td>
</tr>
<tr>
<td>com.android.browser</td>
<td>500kbps</td>
</tr>
<tr>
<td>jp.naver.line.android</td>
<td>Block</td>
</tr>
<tr>
<td>org.mozilla.firefox</td>
<td>Block</td>
</tr>
<tr>
<td>com.android.chrome</td>
<td>2Mbps</td>
</tr>
<tr>
<td>com.google.android.youtube</td>
<td>3Mbps</td>
</tr>
<tr>
<td>wget</td>
<td>5000kbps</td>
</tr>
<tr>
<td>com.skype.raider</td>
<td>Pass through</td>
</tr>
<tr>
<td>mediaserver</td>
<td>3Mbps</td>
</tr>
</tbody>
</table>

Youtube (mediaserver): 3Mbps
Chrome 2Mbps
android.browser: 500kbps
3. Application-Specific Spectrum Scheduling
Application-Specific RB Scheduling

• Default OAI policy (two-round Fair-sharing)
  – Assign maximum average RBs to active UEs
  – Assign remaining RBs to high priority UEs

• Application-Specific Policy
  – Per Application Per Policy, E.g.,
    • Greedy policy: Assign required RBs to UE running specific APP in Round 1
  – Default: Fair-sharing policy
Application-Specific RB Scheduling

- UE1: Running SpeedTest
- UE2: Running iperf (fair-sharing policy)
FlareLTE: Proof-of-Concept of Application-Specific EPC and eNB
Architecture of EPC in FlareLTE

FLARE Switch

FLARE Slice 1 (c-plane)
- docker
- MME
- SP-GW (c-plane)

FLARE Slice 2 (c-plane)
- docker
- MME
- SP-GW (c-plane)

PCI

FLARE Slice 1 (d-plane)
- SP-GW (d-plane)

FLARE Slice 2 (d-plane)
- SP-GW (d-plane)

Slicer slice

Signaling

Southbound API

Data

eNB

PDN
Architecture of FLARE Switch

Commercialized FLARE

FLARE2 PCIe Card

(board designed by Nakao Lab)

x86 Processor

Many Core Processor

Many-core Processors

Control Plane

Data Plane

PCI

intel Xeon E5-2600
Slice Architecture on NPU

LXC: Linux Container on Zero Overhead Linux (ZOL)

LXC:

To X86
Or
To GPGPU
Architecture of eNB in FlarLTE

- Each eNB slice in a Docker instance is isolated and replaceable
- USRP X310 connects to eNB slice via XGBE passthrough technology
- The traffic of different slices is isolated using VLANs by Open vSwitch
FlareLTE’s eNB Optimization

• We assign dedicated data-plane CPU cores to sub frame decoding/encoding threads to reduce processing delay by reducing context-switching

• We use high-frequency AMD CPU since the cost of AMD CPU is only about 30%~50% of that of the same-level Intel CPU
FlareLTE with AMD CPUs

**AMD** Threadripper 1950X

- 16-core/32-thread
- 180w
- 32MB cache
- 3.4GHz
- $868
  (Amazon 2018/3/20)

**Intel** Core i9-7960X

- 16-core/32-thread
- 165w
- 22MB cache
- 2.8GHz
- $1544
  (Amazon 2018/3/20)
FlareLTE in Operation

eNB docker instance

EPC D-plane LXC instance

HSS docker instance

EPC C-plane docker instance

UE
4. Other OAI Activities in UTokyo
Berlin-Tokyo TestBed

Tokyo

Berlin

UE1 - to - UE2 video call

10.100.1.102/24 10.100.2.101/24
Fujitsu-UTokyo Field Test
Field Test Experimental Setup

1. Detect and remove app trailer
2. Record app log file

1. To run flare software
2. To play UE remotely

Application Identification Data

Data Traffic Flow

Internet

192.168.200.31
192.168.110.31

UE1
PC1
UE2
PC2
UE3
PC3

Data collection server

Record signal-level log from eNB

eNB Low Level Data

Intel Server
Fujitsu PRIMERGY

OAI-EPC
ens5
ens6
ens7

OAI-eNB
ens5
ens6
ens7

vlan100
vlan200

USRP

L2SW

FLARE / P4

NAT

1. Detect and remove app trailer
2. Record app log file

Attach app-trailer

PC	1
PC	2
PC	3
Association between Application-level Info (FLARE) and Signal-level Info (eNB)

Air Bandwidth is occupied by UE (rnti=22411, firefox)
Association between Application-level Info (FLARE) and Signal-level Info (eNB)

Resource Blocks are evenly allocated to active UEs in OAI
Thank you!