DEEP PACKET INSPECTION FOR M2M FLOW DISCRIMINATION – INTEGRATION ON AN ATCA PLATFORM

Alexandru BALICA, Cosmin COSTACHE, Florin SANDU, Dan ROBU

"Transilvania" University of Brasov, Romania

Abstract: *In the context of decreased revenues, the mobile operators and technology producers are interested in monetizing the transmission channel used by over the top players. The proposed industrial ATCA integration is suitable for Machine to Machine traffic study, for infotainment or legal survey.*

Keywords: *ATCA, DPI, Data Analytics, Monetization, OTT, Streaming, M2M, PCRF*

1. INTRODUCTION

Deep Packet Inspection (DPI) applications are expected to hit $1.5 B in 2014 (Radisys, 2014). DPI becomes critical for telecom operators in order to add value to the transmission medium used by over the top players (OTT) via services and applications from new mobile hand-held devices. In addition, the traffic is expected to grow even more due to Machine to Machine communication (M2M) in the new Internet of Things, using Quality of Service (QoS), Data Analytics and monetization.

The ATCA platform (Advanced Telecommu & Computing Architecture) is natively suited for high traffic networks due to PIGMG backplane (PCI Industrial Computers Manufactures Group) with a common 40 Gbps "InfiniBand" Switching Fabric behind, useful for very large bandwidth (Bergstrom, 2003; Mellanox, 2009).

For this purpose, we also created a test-simulation environment, useful for development and quick testing before delivering the application to the live platform. By doing this we are following a real-life industrial process for design, testing and service delivery.

2. MEANS AND PURPOSE

2.1 M2M DPI CAPABILITIES. Our purpose is to identify a M2M stream (e.g. DSTP – Data Socket Transfer Protocol) inside a general data capture, alongside HTTP or TCP protocols.

We capture the stream in a file and analyse it or feed the stream directly to the DPI application. For these purposes, we have devised a test environment for debugging and customization.

As test system we are using the ATCA platform from Radisys/Continuous Computing model SH61 40G (fig. 1).

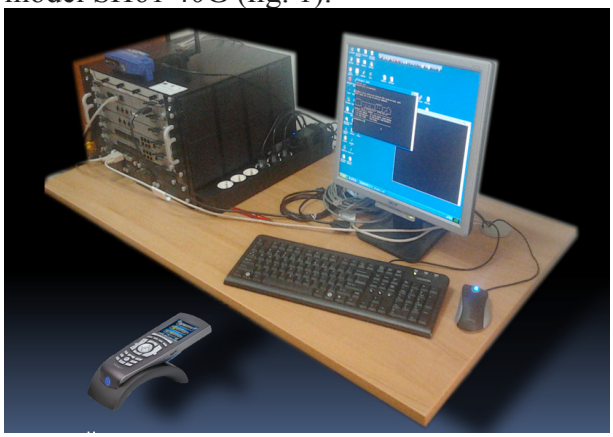

Fig. 1. General view of the ATCA Workbench

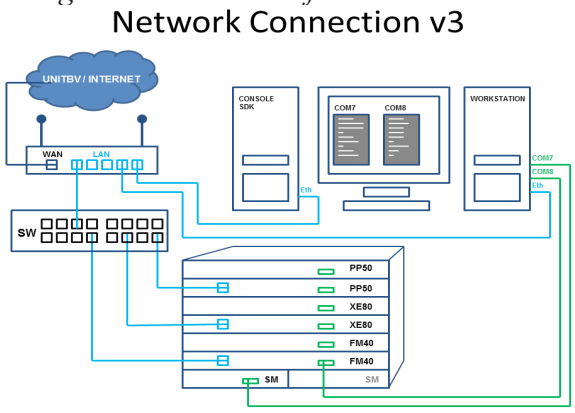

Fig. 2. ATCA Platform Integration – using a router and a multi-port switch for internal (local) and external (remote) access

In the workbench of fig.2, in the platform integrated for our DPI of M2M communications, the 6 available slots are populated as follows: two PP50 boards dedicated to packet processing, two FM40 boards providing advanced switching and two XE-80 computing boards.

2.2 AVAILABLE ATCA INTEGRATION. The physical integration of the ATCA platform is done so that the platform can be remotely accessed for configuration development and testing. The access was devised in order to be able to start the platform only when needed and to eliminate the idle time (Sandu, 2012).

The integration allows remote access for *systolic* process deployment and operation making it also an ideal remote e-Learning tool – with independent sessions possible from different locations, (Machidon et al, 2013).

2.3 CREATING THE TESTING ENVIRONMENT. The simulation environment contains the following components:

- SDK RMI software version – needed for cross-compilation on the RMI / Netlogic / Broadcom XLR 732 MIPS processors used by PP50.
- XLR Simulator for PP50 – used in order to be able to load, run and test the applications.
- *Pretender* – software that provides PP50 external interfaces *gmac* with packets
- Eclipse (C programming IDE – Integrated Development Environment) – used to develop the applications

All the above components are placed together on top Linux Ubuntu OS (Operating System), inside a VMWare image. The Linux OS, the bootloader RMIOS, and Linux loader application being run on a PP50 must all come from the same version of the SDK.

3. SIMULATED ENVIRONMENT

3.1 EXPECTED WORKFLOW ON ATCA PP50. The traffic flow inside PP50 (Continuous Computing, 2011) is simulated on the above-mentioned environment.

The Shared Memory Interface is responsible with the storage of packets. After the Linux Load of applications, `userapp` commands their execution via RMIOS.

Our purpose, in order to be able to parse and inspect packets (DPI), is to identify the masks inside the IP headers as they are stored in the memory.

3.2 APPLICATION LOGIC. Our application, called "PacketClassifier", has two main functions

1. *config_parser* –used to configure the parser engine on the XLR cores, in order to extract packet information (like source and destination IP and port)
2. *process_pkts* – reads each IP header as it arrives and, in case of matching, it proceeds to the classification.

3.3 COLLECTING AND PROVIDING REAL DATA FOR PROCESSING. DSTP on port 3015 represents Nationa Instruments special protocol for the transmissions of measured data is among the most suitable for M2M implementations.

Option 1. Generating a dump *pcap* file using Wireshark that contains a DSTP stream.

For this purpose we have used LabWindows CVI (C for Virtual Instrumentation) built-in Read/Write procedures and we have identified two ways in providing data streams that can be later used by the applications.

The solution is to use a DSTP stream generator and dump the information into a file (for example, we have used already available patterns delivered in from LabWindows CVI – as shown in the following).

Option 2. Using the above-mentioned tool *Pretender* that is provided with the SDK for traffic supply to the *gmac* interfaces:

```
Usage: ./pretender [-p port] -f <tcpdump-
style filter string>

atca@ubuntu $ cd /opt/rmi/fsim_2.0.0/
install/

atca@ubuntu $ sudo ./pretender -p 6001 -i
eth0 -d -f "ether dst 00:1c:25:a2:03:5f"
```

The above command will redirect all packets to gmac0, port 6001, from eth0, if they are addressed (destination) the specified MAC address.

Using this command we can supply real time traffic to the application.

4. APPLICATION DEVELOPMENT IN SIMULATED ENVIRONMENT

4.1 DEVELOPING AND TESTING REAL APPLICATIONS SUITABLE FOR M2M OVER PP50.

Using Eclipse IDE for the initial C programming of services, we benefit from various DPI templates and examples. These examples are then upgraded for practical use in packet filtering according to our needs.

4.2 DATA STREAM INPUT. To validate the applications, they were tested using stream inputs. The data stream is redirected by *Pretender* from the eth0 traffic based on destination MAC (i.e. remote workstation). After that, we launched the application on this streaming input from the local interface eth0. As *Pretender* is already running, the application connects to the stream as soon as it is started, and the `Accepted Connection` is announced.

```
atca@ubuntu $ ./fsim -f boot1
-F0x10000000=/home/atca/workspace-rmi/
PacketClassifier2/PacketClassifier2 -C
gmac0.serverip=192.168.81.128 -C gmac0.
serverPort=6001 -C gmac0.mode=0
```

To test this, we can launch the `ping` command from the local station to various valid destinations and the ICMP (Internet Control Message Protocol) messages associated are visible on the *Pretender* and on the application that is decoded.

4.3 DATA FILE INPUT. By providing the application with a *dumpfile* for analysis we are also validating the off-line functioning mode of the integration.

For monitoring reasons, we had to launching the application first. It would halt until the input connection with *Pretender* is established, but in this way we ensure that the application would process when we want:

```
atca@ubuntu $ ./fsim -f boot1
-F0x10000000=/home/atca/workspace-rmi/
PacketClassifier2/PacketClassifier2
atca@ubuntu $ elfload -a 0x10000000
atca@ubuntu $ userapp
```

From this moment on, there are several ways to feed off-line information to the application. As *Pretender* is using a default file `rxPktFilegmac0` for provisioning the gmac0 interface to the application our efforts consist in populating this data file.

A. We can test the transmission from file using the test tools provided with *Pretender*. The tool *gen_pktfile* will place test traffic into `rxPktFilegmac0` that is being read by the *Pretender*, like in the following:

```
atca@ubuntu $ ./gen_pktfile -o
rxPktFilegmac0 -s 100 -n 10
```

Since *Pretender* is launched and the application is loaded we will see the processed messages in the application's output:

```
[process_pkts]:    cpu_id=0,    Processing
Packets...
[process_pkts]: received message <size=1,
code=0, stid=96 msg0=80006a0001000800
[process_pkts]: received message <size=1,
code=0,   stid=96   msg0=80006a0001000800,
msg1=0
              00: 00 01 02 03 04 05 06 07
              08: 08 09 0a 0b 08 00 45 46
              16: 00 54 00 40 1b e8 40 06
              24: 83 6d c0 a8 01 01 cb 1f
              32: 4e 00 0d b7 31 58 a3 5a
              40: 25 5d 05 17 58 e9 5e d4
              48: ab b2 cd c6 9b b4 54 11
              56: 0e 82 74 41 21 3d dc 87
[process_pkts]: received message <size=1,
code=0,   stid=96   msg0=80006a0001000e60,
msg1=0
              00: 00 01 02 03 04 05 06 07
              08: 08 09 0a 0b 08 00 45 f5
              16: 00 54 00 40 17 fc 40 06
              24: cf 8a c0 a8 01 01 40 3f
              32: 90 00 f3 c4 c3 77 9a 64
              40: b2 be b5 d1 db 20 c6 35
              48: 9d d6 0e b4 d3 b3 f2 5f
              56: d7 e1 5b b1 b0 a9 5d 63
[process_pkts]: received message <size=1,
code=0,   stid=96   msg0=80006a00010014c0,
msg1=0
```

B. Real life scenario involves using a *dumpfile*. We collected a `pcap` trace `dumpfile.pcap` that contains DSTP traffic as shown in paragraph 3.2. This *dumpfile* must be used as input for *Pretender* that will re-format it over gmac0 into a *fsim* compatible format. In order for *Pretender* to use this *dumpfile*, it must be presented as default `rxPktFilegmac0` file so we had to rename the `dumpfile.pcap` to `rxPktFileGmac0`.

A valid way to send the stream from the *dumpfile* towards the application is to use the *Pretender* as interceptor for data but generate the stream using an utility like `dumpcap`, `tcprreplay or bittwist` (some of them might require installation e.g. `sudo apt-get install tcpreplay`).

We have successfully used `bittwist` to send the `pcap` file to *Pretender*. The only constraint we have met is that we need to re-format the `pcap` file before, using `tcprewrite` like below:

```
atca@ubuntu $ tcprewrite -i dumpfile.
pcap -o test1.pcap --dlt=enet --enet-
smac=00:0C:29:42:F8:2A --enet-
dmac=00:0C:29:42:F8:2B
```

With this command we are adding test MAC addresses through conversion. After this step we obtained the `pcap` file `(test1.pcap)` to be transmitted. The full execution cycle in the development simulated integration, see Fig. 3, becomes:

1. *Pretender* start on eth0 with MAC address filtering on the values added in `test1.pcap`.

```
tca@ubuntu $ sudo ./pretender -p 6001 -i
eth0 -d  "ether dst 00:0C:29:42:F8:2B"
```

2. *fsim* application load with connection with *Pretender* over port 6001

```
atca@ubuntu $ ./fsim -f boot1
-F0x10000000=/home/atca/workspace-rmi/
PacketClassifier2/PacketClassifier2 -C
gmac0.serverip=192.168.81.128 -C gmac0.
serverPort=6001 -C gmac0.mode=0
```

3. Send packets to eth0 using `bittwist`:

```
atca@ubuntu $ sudo bittwist -i eth0 test1.
pcap
[sudo] password for atca:
sending packets through eth0
trace file: test1.pcap

4061 packets (726524 bytes) sent
Elapsed time = 11.861160 seconds
```

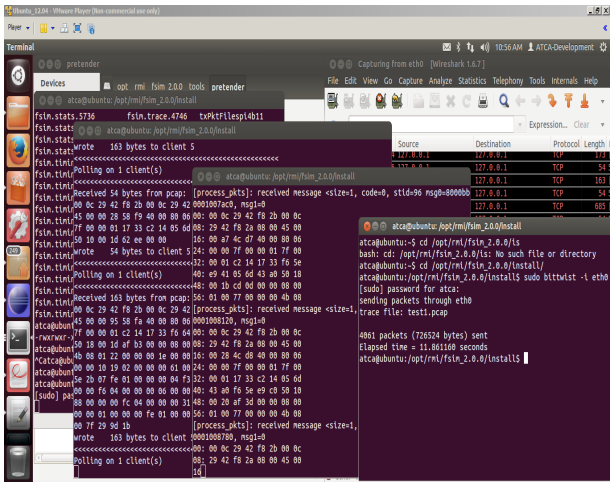

Fig. 3. Application execution on simulated environment (Pretender, fsim userapp, bittwist and Wireshark monitoring)

4. Wireshark monitoring on eth0 will provide visual confirmation.

5. PACKET PROCESSING ON THE ATCA 40G PLATFORM

5.1 EXPECTED WORKFLOW AND EXAMPLES. After development, testing and validation in the simulated environment, we took the following steps in order to execute the application on the real equipment.

Due to the remote connection, the application must be copied from the simulated environment (e.g. development can be done on any PC) to the workbench computer (the "console" connected to the ATCA platform of fig. 2). Only after that, the application can be copied via tftps to the PP50 blade and be executed. The workbench "console" is remotely controlled via TeamViewer.

1. Building the Eclipse project (Build option in Eclipse) in the VMWare image (in our case, it results the *PacketClassifier* file)

2. Local Copy of the software file (*PacketClassifier*) on the computer (e.g. a PC) hosting the simulated environment. The local copy is done via VMWare Option for Shared Folder, via `wmare-config-tools.pl`.

3. Remote Copy to the ATCA workbench computer via TeamViewer File Transfer.

4. Local Copy on ATCA Platform via tftpc – Tftpd32 (Sandu, 2013).

5. Load and execute application on ATCA PP50 in the XLR actual environment.

5.2 INTERFACES CONFIGURATION. Configuring the interface for remote copy means assigning on gmac0 (eth0 under Linux) an IP valid for communication: 192.168.1.147.

As we followed the industry standard, we assigned IPs in sub-domains, separating traffic from management.

Our IP definitions presented below represent administration IPs, so they are going to use default assignments under 192.168.1.X:

```
PP50-0 $ ifconfig -i gmac0
PP50-0 $ Starting Network interface
"gmac0"
Can not get kv entry sysid.
Can not get kv entry shelfid.
Value of key hwaddr = 0x45
Starting dhcp...
Got DHCP response,waiting for IP
assigned...
```

```
IP          : 192.168.1.147
Tftpserver  : 192.168.1.1
Bootfile    : []
netmask     : 255.255.255.0
Default GW  : 192.168.1.1
DNS         : 192.168.1.1
```

To test the validity of gmac0 we could also verify the connection from gmac0 after loading a (default) Linux image:

```
PP50-0 $ dload pcmcia_1 1 /pp50-linux-rmi16-v2.6.3r00
PP50-0 $ userapp -f
```

PP-50 has following IP on eth0 under Linux (i.e. this means the physical gmac0 of the PP-50): 192.168.1.148 and the MAC address is: 00:02:bb:52:9c:80 (00-02-bb-52-9c-80).

We have let Linux DHCP obtain the IP on this interface (i.e. eth0) and tested successfully the connection to the Workstation (192.168.1.179):

```
[root@localhost ~]$ dhcpcd eth0

[root@localhost ~]$ ping 192.168.1.179
PING 192.168.1.179 (192.168.1.179): 56 data bytes
64 bytes from 192.168.1.179: icmp_seq=0 ttl=128 time=0.870 ms
64 bytes from 192.168.1.179: icmp_seq=1 ttl=128 time=0.219 ms
```

Our goal was to check that gmac0 set-up was correct and outside access (in this case to the Workstation) is possible.

We will expect the same behaviour after loading the application, with the gmac0 being correctly configured: eth0 (gmac0) will receive traffic that is sent to it from the Workstation.

5.3 PROGRAM COPY AND EXECUTION. Copying the build (PacketClassifier) command tftpc:

```
tftpc -s 192.168.1.179 -b 1024 -f PacketClassifier
```

We are using Tftpd32 freeware in order to set up a tftp server. The source IP is the IP in the local network (switch-router) of the Workstation (172.168.1.179)

From the PP50 we are initiating a tftpc (copy) in memory of the build.

```
PP50-0 $ tftpc -s 192.168.1.179 -b 1024 -f PacketClassifier
Downloading [PacketClassifier].
Server IP : 192.168.1.179
PP50-0 $ tftpc stall; Check network setup.
Bytes downloaded: 1493018
tftpc: download done. Size [1493018] @ Addr [0x120000000]
PP50-0 $ elfload -a 0x120000000
PP50-0 $ userapp
```

For any subsequent load, it was actually used directly the `userapp` command:

```
dload pcmcia_1 1 /pp50-linux-rmi16-v2.6.3r00
userapp -f
```

5.3 APPLICATIONS OUTCOME. Based on the criteria selected in the masks, TCP and UDP traffic can be separated – with all the relevance that comes from the upper OSI layers. HTTP or, in our case, DSTP for M2M services can be further analysed.
Below is an example for simple IP filtering:

```
// extract the destination IP
dest_ip = (recv_pkt->data[30]<<24)|(recv_pkt->data[31]<<16)|(recv_pkt->data[32]<<8)|(recv_pkt->data[33]);

printk("Destination IP: %d.%d.%d.%d \n",
 (int) ((dest_ip >> 24) & 0xff), (int) ((dest_ip >> 16) & 0xff),
 (int) ((dest_ip >> 8) & 0xff), (int) ((dest_ip >> 0) & 0xff));
// extract the source IP
src_ip = (recv_pkt->data[26]<<24)|(recv_pkt->data[27]<<16)|(recv_pkt->data[28]<<8)|(recv_pkt->data[29]);
printk("Source IP: %d.%d.%d.%d \n",
 (int) ((src_ip >> 24) & 0xff), (int) ((src_ip >> 16) & 0xff),
 (int) ((src_ip >> 8) & 0xff), (int) ((src_ip >> 0) & 0xff));
```

CONCLUSIONS

As DPI is becoming a "must have" capability for the Telecom operator, dedicated ATCA platform integrations are to become more common. The flexibility of the platform is somehow restricted by the complexity of configuration and its industrial cantered applicability.

The ATCA integration we proposed can be used to simplify operational work on this type of platform and meet the demand for specialization by using it in a systolic mode of operation, both for development and for didactical purpose.

The new PCRF (Policy and Charging Rules Function) component available in LTE networks is available exactly because of powerful data processing that comes with ATCA implementations.

Our study, practical integration and development show the relative simplicity of programming traffic discrimination as long extremely powerful packet processors are available.

Statistics, control, filtering, resource allocation or legal survey can be done using this technique, empowering the operators and providing a much needed leverage in front of OTT.

ACKNOWLEDGMENT

This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD), ID134378 financed from the European Social Fund and by the Romanian Government.

BIBLIOGRAPHY

1. Radisys (2014). Partnering For DPI Deployment. *FierceMarkets custom publishing.* Available: http://go.radisys.com /rs/radisys/images/ebook-atca-partnering-for-dpi-deployment.pdf [last accessed: April 2014].
2. Bergstrom, E. (2003): Advanced TCA: A Force of One in Telecom & Datacom Applications. *Crystal Cube Consulting*.
3. Sandu, F., Balica, A.N., Robu, D.N., Svab, S.R. (2012): Remote Access to an Advanced Telecommunications Platform for Educational Purpose, *Proceedings of the 7th International Conference on Virtual Learning (ICVL),* Bucharest University Press, ISSN 1844-8933, pag. 413-420.
4. Machidon, O.M., Sandu, F., Balica, A.N., Robu, D.N., Cazacu, V. (2013) Remote Project Integration on an ATCA Platform, *Proceedings of the 11th International Conference: Networking in Education and Research*, *RoEduNet* ISSN: 2068-1038, Print ISBN: 978-1-4673-6114-9 – *ISI Proceedings.*
5. Sandu, F., Costache, C., Balan, T.C., Balica, A.N. "Packet Processing on an ATCA 40G Platform" 2013, Proceedings of the 4th International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics (MACRo2013), Scientia publishing house, 2013, ISSN: 2247 – 0948, pp. 227-238 / 239-250 / 251-258; 4-5 October, 2013, Tirgu Mures, Romania.
6. Continuous Computing (2011) "FlexPacket ATCA PP50 Packet Processor", *User Manual*, CC06786-11B