Poster – RECO: A Reconfigurable Core Network for Future 5G Communication Systems

Chia-Han Wu
Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
chiahan.cs04g@g2.nctu.edu.tw

Wei-Jen Chen
Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
batx6.cs00@g2.nctu.edu.tw

Jyh-Cheng Chen*
Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
jcc@cs.nctu.edu.tw

ABSTRACT

It is envisioned in the future that not only smartphones will connect to cellular networks, but all kinds of different wearable devices, sensors, vehicles, home appliances, VR headsets, robots, will also connect to cellular networks. Because the characteristics of these devices differ largely, people argue that future 5G systems should be designed to elastically accommodate to the different users. In this paper, we propose a reconfigurable core network called RECO that demonstrates how to implement customized virtual core network entities efficiently to suit for users with different characteristics. We then build a testbed to verify our proposed RECO architecture. Finally, we will open source our RECO in the near future so that the research community can take benefit out of it.

KEYWORDS

5G; Reconfigurable Networks; Network Function Virtualization; System Design and Experimentation

1 INTRODUCTION

The future 5G communications system is envisioned to transform from a human-oriented system into an *Internet of Things (IoT)* system. This new system will connect tens of billions of devices ranging from mobile phones, tablets, home appliances, wearable devices, VR headsets, sensors, vehicles, robots, and so on, making our lives smarter and more convenient. The NGMN 5G white paper shows new use cases that will emerge beyond 2020 [8]. The use cases range from delay-tolerant applications to extreme real-time communications, from high user mobility to pedestrian speed mobility, from best effort network applications to ultra-reliable ones, and from an average connection density network for smartphones to an extremely high connection density, low-cost, long-range, low-power network for sensors.

Due to the diverse characteristics of different use cases, the future 5G cellular networks should be designed to elastically accommodate such a heterogeneous environment. In other words, a 5G core network should be able to flexibly customize itself to serve different use cases. To achieve this goal, we propose **Re**configurable **Co**re (RECO) that can flexibly support different types of users with different characteristics. In addition, we have built a prototype and tested it with commercial User Equipment (UE) terminals and a real LTE base station. Please refer to our demo video [1]. We plan to release the whole source code of RECO to [2] soon. In the past, the cellular core network was very expensive and not easy to access the source code. With Object-Oriented Programming (OOP) design in our RECO testbed, however, researchers not only can get the source code for free, but also can add their own modules easily. For people doing research in this area, they usually could only conduct mathematical analysis and simulation to verify their ideas. In contrast, our RECO testbed provides a low-cost way for researchers to implement new ideas in a real core network environment. Professors teaching courses in cellular core networks can also design labs for students, which is not easy to achieve without open-source code with modularized design.

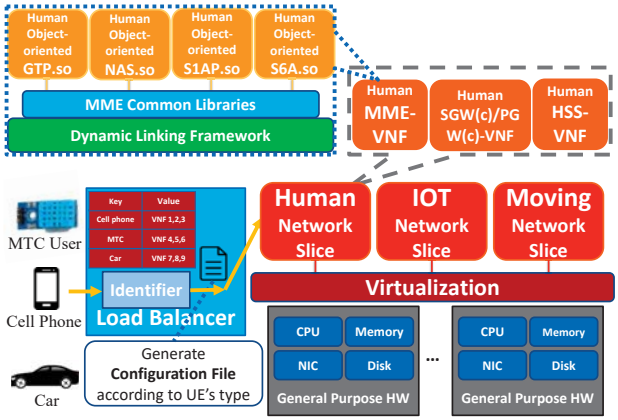

Figure 1: RECO Architecture

2 RECONFIGURABLE CORE (RECO)

To build a flexible 5G core network, we propose RECO and the architecture is shown in Fig. 1. RECO is designed based upon a virtualized core network environment using Network Function Virtualization (NFV) and has three key components:

(1) Modularized virtual network functions (VNFs) designed by OOP language: We split the code for virtual network functions (virtual network entities) into modules and compile them into shared libraries (.so). We then separate the modules into two groups: (i) ***common modules*** which are the same among different types of users, and (ii) ***customized modules*** which differ between different

*Corresponding author: Prof. Jyh-Cheng Chen

MobiCom '17, October 16-20, 2017, Snowbird, UT, USA.
 ISBN 978-1-4503-4916-1/17/10.
DOI: http://dx.doi.org/10.1145/3117811.3131257

types of users. As shown in Fig. 1, the Mobility Management Entity (MME) common libraries are the common modules and GTP.so, NAS.so, S1AP.so, and S6A.so are the customized modules. Different customized VNFs on the same machine can share the same copy of common modules. For example, a human MME VNF, an IoT MME VNF and a vehicle MME VNF can all share the same copy of common modules. On the other hand, customized modules differ between different user types and should be implemented in OOP languages. The major benefit of implementing customized modules like this is that it can highly reuse existing code and promote the process of creating new customized VNFs. For example, suppose we have already implemented a human MME VNF. We can directly inherit most of the classes within the human MME VNF and just override particular mobility-related member functions to build a high-mobility MME VNF for vehicles.

(2) Dynamic linking framework which links customized modules during run-time: For each virtual network entity, there is a dynamic linking framework inside it. The framework's major job is to link customized modules during run-time according to the configuration file provided by the identifier to form a customized virtual network entity. The reason we choose to use dynamic linking is because it highly increases the flexibility to create a new network entity. When a new virtual network entity is needed, all we need is to create new corresponding customized modules for the particular type of users. The dynamic linking framework then will compose everything together to form a new network entity. This is somewhat similar to adding a plugin into the Chrome browser to generate a customized browser.

(3) An identifier which generates a configuration file to form customized virtual network entities: When a new user first tries to attach to the core network, the identifier inside the load balancer will identify the user's type by using some mechanism such as parsing an ***user type tag*** in the attach request packet. If this kind of user type has never attached to the core network before, the identifier will generate a ***configuration file*** including information about what customized modules are needed to form particular virtual network entities. The identifier then will pass it to the dynamic linking framework. Accordingly, the dynamic linking framework will compose these modules together and form customized VNFs to serve the type of users. The identifier also records a ***hash table*** with user types as the key and index of corresponding VNFs for particular user types as the value. In this manner, the load balancer can forward subsequent packets of the same user type to corresponding VNFs according to the hash table.

3 RECO MME AND RECO TESTBED

We built a reconfigurable virtual MME to demonstrate the proposed RECO architecture. It is modified from the MME of openair-cn [6], a simple core network developed by EURECOM [7]. We separated the highly bundled MME executable linked with many static libraries in openair-cn into a dynamic linking framework linked with shared libraries. Fig. 2 shows the architecture of our RECO MME. It is composed of four main components:

(1) Common modules: We modified the static libraries used in openair-cn's MME into two kinds of shared libraries: (i) *load-time shared libraries* shared among different types of users, and (ii)

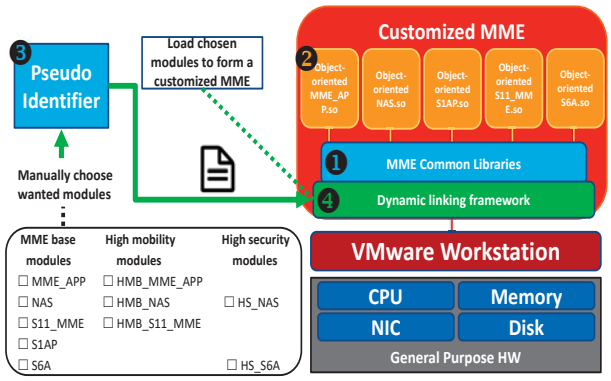

Figure 2: RECO Testbed

run-time shared libraries differing between types of users. The *load-time shared libraries* are common modules shared among different customized MMEs. When the MME executable (the dynamic linking framework) is executed and loaded into memory, these common modules will also be loaded into memory. Using common modules can reduce the disk and memory usage.

(2) Customized modules: We built five modules MME_APP, NAS, S1AP, S11_MME and S6A inside openair-cn from static libraries into *run-time shared libraries*. Building these customized modules into *run-time shared libraries* enables the MME to load and unload shared libraries during run-time. By doing so, the dynamic linking framework can load different customized modules according to the configuration file and form a customized MME. In addition, the source code inside these five modules are all refactored from C to OOP language C++. This was done by composing related functions and variables inside each module into classes. By doing so, a programmer can use the capabilities of OOP such as inheritance, override, polymorphism, etc. to extend the existing modules into new customized modules in a more efficient manner. This highly promotes the process of development.

(3) Pseudo Identifier: Currently, we simply implemented a checkbox list as a pseudo identifier to mimic the behavior of a real identifier as shown in Fig. 2. Programmers can choose particular modules for a type of user manually through the checkbox list. The pseudo identifier then will generate the corresponding configuration file and pass it to the dynamic linking framework.

(4) Dynamic linking framework: The dynamic linking framework is used to link customized modules at run-time according to the configuration file provided by the pseudo identifier. Its main functionalities are to provide a standard interface for customized modules, parse the configuration file, load and initialize corresponding customized modules according to the configuration file, and help resolve dependency relationships among customized modules.

4 EVALUATION

4.1 Verify Correctness of RECO

In order to verify the correctness of our RECO MME, we connected it with real commercial UEs and a real commercial LTE small-cell eNB along with S-GW, P-GW and HSS provided by openair-cn. Because the commercial UEs can successfully connect to the

Table 1: Testbed comparison

Testbed	Open source	Completeness	Status	Reduce disk&memory	Flexibility	OOP Design
Open5GCore [3]	Very Expensive	✓	Active	✓	✓	
OpenEPC [4]	Very Expensive	✓	Active	✓	✓	
nwEPC [5]	✓		Frozen			
openair-cn [6]	✓		Active			
RECO MME [2]	✓		Active	✓	✓	✓

Table 2: Performance Overhead for RECO MME

	MME Setup Time	S1 Setup Time	Initial Attach Time (1 PC)	Initial Attach Time (3 PCs)
RECO MME	57.64536 ms (+15.72%)	0.4717692 ms (+3.23%)	12.996 ms (+8.62%)	15.6608 ms (+4.19%)
openair-cn	49.81621 ms	0.457 ms	11.96421 ms	15.0304 ms

Internet through a commercial LTE eNB and our testbed, our RECO MME is implemented correctly.

4.2 Benefits of using RECO

Compared to other testbeds shown in Table 1, RECO has the benefits of reducing memory and disk space and can flexibly load and unload customized modules during run-time. This is because we build our RECO MME by using dynamic linking technologies just as commercial products OpenEPC and Open5Gcore do. However, OpenEPC and Open5Gcore are too expensive for most universities. Besides, as far as we know, RECO is the only testbed implemented in an object-oriented code structure. By doing so, developers can inherit many existing classes and override member functions with customized functionalities which improve the development process of building customized virtual network entities.

4.3 Performance Overhead of RECO

Although using dynamic linking and C++ programming have the benefits described in Sec. 4.2, there is performance overhead comparing to using static linking and C. To measure the performance overhead, we measured MME Setup Time, S1 Setup Time, and Initial Attach Time for our RECO MME compared to the original openair-cn MME. Table 2 shows that RECO MME takes 15.72% more time than openair-cn MME to finish initialization. This is because RECO MME needs additional linkage time on every execution due to the use of dynamic linking. Although this overhead seems excessive in proportion, it is actually only less than eight milliseconds, which is acceptable. As for S1 Setup Time, RECO MME costs only 3.23% more than the original openair-cn. For Initial Attach Time, we installed each core network entity in a virtual machine (VM) and deployed these VMs (RECO MME VM, openair-cn SPGW VM, openair-cn HSS VM) on 1 PC and 3 separate PCs. The results show that RECO MME costs only 8.62% and 4.19% more, respectively. The reason that the overhead decreases on 3 separate PCs is because the transmission delays between the 3 PCs reduce the impact of dynamic linking and C++ overhead. In a real cloud environment, there might be more than 3 physical machines. Thus, the delay caused by the distance between UE, eNB, and other network entities would make the dynamic linking and C++ overhead negligible.

5 FUTURE PLAN

We plan to release the whole source code of RECO to [2] soon. Besides, we will provide complete documentation including how to install RECO, how to add a new customized module, and how to program a real SIM card to connect to our core network. Thus, researchers can easily implement and test their proposed algorithms in a real testbed. For example, if a researcher propose a new paging algorithm, traditionally the researcher could only simulate the new algorithm. With RECO, the researcher can now inherit most of the classes developed by us and only modify paging-related member functions to form new customized modules. The dynamic linking framework then will automatically link these new customized modules together to generate a new customized MME with the newly proposed paging algorithm inside it. This would greatly benefit the research community.

ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Science and Technology of Taiwan under grant numbers MOST 106-2221-E-009-046-MY3, MOST 106-2221-E-009-047-MY3, MOST 106-2218-E-009-016 and MOST 105-2218-E-009-008. We would also like to thank the people who contributed in different ways to this research, including Hong-Shiang Lan, Yu-Sen Hsueh, Jia-Ru Li, You-Min Lin, Yi-Hua Wu, Fu-Cheng Chen, Chia-Wei Chang, Ping-Fan Ho, Bo-Jun Qiu, Yi-Hao Lin, and Chia-Che Hsu.

REFERENCES

[1] https://youtu.be/6iuCq350LYM.
[2] https://github.com/RECONet/RECO.
[3] http://www.open5gcore.org/.
[4] http://www.openepc.com/.
[5] https://sourceforge.net/projects/nwepc/.
[6] https://gitlab.eurecom.fr/oai/openair-cn.
[7] http://www.eurecom.fr/en.
[8] NGMN Alliance. NGMN 5G white paper. *Next Generation Mobile Networks, White paper*, 2015.