Towards the Virtualization of Transport-level Functions and Protocols

El-Fadel Bonfoh, Samir Medjiah, Christophe Chassot, José Aguilar

▶ **To cite this version:**

El-Fadel Bonfoh, Samir Medjiah, Christophe Chassot, José Aguilar. Towards the Virtualization of Transport-level Functions and Protocols. 7th IEEE International Conference on Smart Communications in Network Technologies (SACONET'18), Oct 2018, El Oued, Algeria. 10.1109/SaCoNeT.2018.8585578 . hal-01959194

HAL Id: hal-01959194

https://hal.insa-toulouse.fr/hal-01959194

Submitted on 18 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards the Virtualization of Transport-level Functions and Protocols

El-Fadel Bonfoh [1,3], Samir Medjiah [1,2], Christophe Chassot [1,3], Jose Aguilar [4]

[1] CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France

[2] University of Toulouse, UPS, LAAS, F-31400 Toulouse, France

[3] University of Toulouse, INSA, LAAS, F-31400 Toulouse, France

[4] CEMISID, Facultad de Ingeniería, Universidad de Los Andes, Mérida, Venezuela

{efbonfoh, medjiah, chassot} @ laas.fr, aguilar@ula.ve

***Abstract*—The Transport layer of OSI and TCP/IP models provides all necessary services for end-to-end communication between application processes. There are a huge amount of works and propositions of Transport level protocols and services to satisfy applications requirements. Unfortunately, the vast majority of applications refer only to TCP for reliable and ordered services or to UDP for unreliable and low latency services. This is due to the fact that the deployment of all new Transport protocol proposal is mainly hampered by (1) the poor socket API exposed by the Transport layer to applications, (2) the introduction of middleboxes within the Internet and (3) the tedious work required to modified Operating System kernel. At the same time, the development of network functions virtualization opportunities is growing in the world of carrier networks and more generally. In this paper, after a survey on Transport protocols deployment issues, we present a novel approach to realize the effective deployment of Transport protocols and services by leveraging virtualization principles. At last, we present a new way to efficiently manage Transport functions and to dynamically build Transport services through a graph-based protocol model.**

Keywords — Transport protocols, TCP/IP model, Network Function Virtualization, Graph-based models.

I. Introduction and background

The Internet was designed around a layered architectural model, where each layer plays a very specific role in the proper functioning of the Internet and its components. The standard reference model is the OSI model [1] that consists of 7 layers: the Physical layer, the Data Link layer, the Network layer, the Transport layer, the Session layer, the Presentation layer and the Application layer. Judged too theoretical and unsuitable to the specific context of the Internet, the OSI model was superseded by the more practical TCP/IP model [2], which proposes an architecture in 4 layers: the Network Access layer, the Internet layer, the Transport layer and the Application layer. From the layers of OSI and TCP/IP models, there is one whose role and position are globally invariant from one model to another, it is the *Transport layer*.

The Transport layer, often considered as a hinge layer between the high and low layers, provides the services necessary for end-to-end communication between processes [2]. In the TCP/IP standard, two main protocols have emerged at the Transport layer: TCP [3] and UDP [4] protocols. TCP protocol is used to provide a reliable, ordered and connection-oriented services, while UDP protocol is connectionless and guarantee neither order nor reliability. Unfortunately, it is well known that TCP and UDP protocols are insufficient to face the requirements of critical applications (Interactive video conferencing, Virtual Reality, among others) and in addition, are not suitable to several network contexts. For instance, TCP is known to suffer from limitations in the *Wide Wireless Area Networks* (WWAN), such as Satellite network [5], or more recently, the *Tactile Network* environments [6], where latency have to be under one millisecond.

Nevertheless, despite their limitations, TCP and UDP protocols remain systematically and massively used by the various networking stakeholders, notably the application developers. This situation leads to an underutilization of the opportunities of the Transport layer. Indeed, an optimal Transport layer is supposed to provide the best possible performances taking into account both quality of service (QoS) required by the applications, and the capabilities of the underlying network. A large number of research works have been done to deal with the Transport layer issues, by considering applications requirements and/or network state. These works take two major directions of research: (1) proposing a ***single protocol*** by improving or adding features to the existing TCP protocol, or by developing a new protocol from scratch or on top of UDP protocol; and (2) redesigning the ***entire Transport layer*** such that to enable *service requests* by applications instead of a particular protocol invocation, and to promote the Transport layer evolution through the easy integration of protocol components.

TCP has known a huge number of modifications and extensions since its conception in the 70's. Most of these extensions aim to improve existing TCP options or mechanisms like *Congestion Control* or *Selective Acknowledgement* (SACK) [7]-[12]. The rest of TCP extensions aim at adding to it new features like security,

multi-streaming, etc. For example, Multipath TCP (MPTCP) [13] is an extension of the TCP protocol that provides it a multipath capability by leveraging the ability of today's devices to having multiple network interfaces (Wi-Fi, 3G, LTE, …).

Rather than extending TCP features, there is a lot of works that propose to build new protocols from scratch. The Stream Control Transmission Protocol (SCTP) [14] is a reliable Transport protocol, which operates on top of a connectionless packet network-level protocol, such as IP, and mainly offers multistreaming and multihoming services. In order to behave fairly with TCP flows and leverage UDP protocol efficiency, the Datagram Congestion Control Protocol (DCCP) [15] was developed. It is an unreliable Transport protocol that provides congestion control features to applications.

All previously mentioned protocols are monolithic programs hardly configurable, inextensible, and tedious to update. To overcome these issues and provide QoS capability to Transport protocols, works on configurable and reconfigurable protocols were carried out during the last decade and conducted to a lot of propositions, like the Enhanced Transport Protocol [16]. ETP is a modular Transport protocol QoS-oriented aimed at implementing behavioral and structural adaptation strategies based on the network environment conditions, and guided by the application requirements [17]. More recently, Google has proposed the Quick UDP Internet Connections (QUIC) [18], built on top of UDP, and aiming to reduce latency compared to that of TCP.

Apart from the QUIC protocol supported by Google and MPTCP which is based on TCP, any new proposals for Transport protocols, including those mentioned in the previous paragraphs, have not been massively adopted on the Internet. This lack of deployment is due to two main factors: (1) the backbone of the Internet is subject to an increasing integration of ***middleboxes*** capable of inspecting a packet to the L4 level and rejecting any unauthorized protocol, and (2) the limited Application Programming Interface, called ***socket API***, exposed by the Transport layer to applications. Indeed, socket API ties applications to a specific protocol, so that applications need to be rewritten to support any new protocol. To overcome this issue, [19, 20, 21] argue that the whole Transport layer has to be redesigned, such that the application no longer chooses the Transport protocol it wants to use, but instead invoke the desired *Transport services* and let the layer chooses the adequate protocol (or Transport components) to provide requested services. The main drawback of the majority of these approaches is that, to no more use socket API, applications need to be rewritten to take into account the new proposed framework. Furthermore, to be effectively and efficiently used, the Operating System developers (Windows, Linux, …) must *open* their systems to permit the integration of these frameworks in the OS kernel. This is far from being obtained due to the tedious work required to update an operating system. This situation contributes to the problem of new protocols deployment.

More recently, new paradigms such as Network Function Virtualization (NFV) [22] and Software Defined Networking (SDN) [23] arrived at maturity. In this paper, we argue that by leverage the virtualization concepts as ETSI defined it, we are able to dynamically and timely deploy Transport components on the appropriate network nodes in different contexts (e.g. IoT, Cloud, etc.). Indeed, because each Transport protocol can be seen as an assemblage of *functions*, we define an architecture allowing to directly place the appropriate functions within a network entity (computer, router, switch, etc.). In the first stage of our works, our main focus is to package the Transport functions as virtualization containers (Docker, LXC, etc.) even if they are complex or simple. In the second and last stage of our works, for the simple Transport functions, we will wrap them as Linux kernel modules.

Over the last decade, there are more and more works that took advantage of the virtualization principles to improve networking, by facilitating packets processing function deployment and upgrading. In [24], B. Pfaff et al. present Open vSwitch (OVS), a network switch specifically built for virtual environments. OVS is mainly used to steer the traffic between virtual machines within hypervisors and with the external world. More recently, Z. Niu et al. present *Network Stack as a Service* concept [25], a new way for cloud providers to offer networking features to their tenants inside a container. In this paper, we envision a novel way to manage and deploy network functions at the Transport layer.

The rest of the paper is organized as follows. In section , we detail the reasons for the non-deployment of any Transport protocol other than TCP and UDP protocols. Next, we introduce the NFV approach in section III. Section IV presents the *Virtual Transport Protocol* concept and the aspects related to it. Finally, we end the paper by presenting future works.

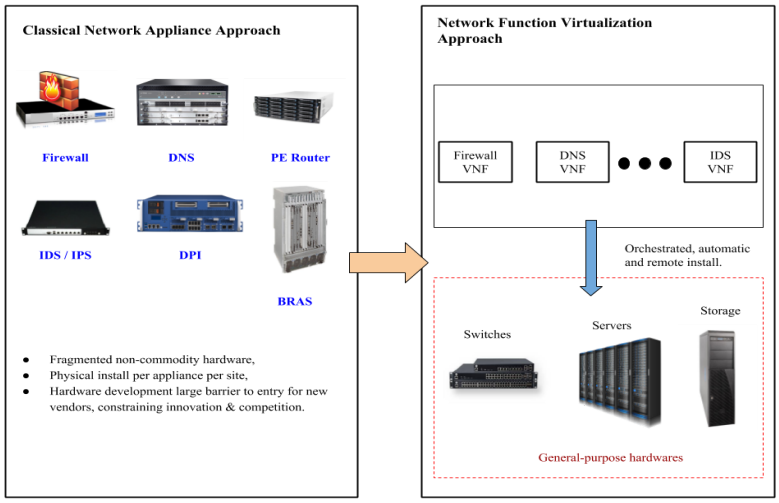

Fig. 1: A Network Function Virtualization Vision [22]

II. Main Limitation of Transport Layer: Deployment

To provide Transport features beyond those of TCP and UDP protocols, a plethora of propositions has been made by researchers. Unfortunately, most of these propositions have not been widely deployed. This is due to three main reasons.

Middleboxes. Initially designed on an end-to-end principle, at least from level L4 to level L7, the Internet has seen the massive introduction of new network devices, called middleboxes (NATs, firewalls, etc.), with the aim to respond to a number of new requirements, such as security, broke the end-to-end semantic. Therefore, to be used, a protocol must not only be integrated and known by the endpoints (i.e. the final hosts) but also, be supported by the middleboxes on its way, so that it can cross the network without failure: this hampers and complicates the deployment of new Transport solutions.

Socket API. In order to provide Transport-level services to applications, the Socket API was developed in 1983. This API was designed so that the application programmer explicitly specifies the desired Transport protocol. This has the disadvantage of, on the one hand, limits the Transport services (provided to the applications) to those of the protocol chosen at design time, and on the other hand, requires the rewrite of the application to support a new protocol.

Vicious circle. This issue is well described in [26]. Application programmers are unwilling to use a new protocol that is unlikely to work end-to-end; OS vendors will not implement a new protocol if application programmers do not express a need for it; middleboxes vendors will not add support if the protocol is not in popular operating systems; the new protocol will not work end-to-end because of lack of support in middleboxes.

We argue that in the vicious circle described above, OS vendors are the key actors and by enabling new protocol support in their operating systems, the adoption of any new protocol by other actors can be effective. Because it is tiresome for OS vendors to integrate a new protocol in the kernel of operating systems, many efforts have been made to implement it in the user space of operating systems [27, 28]. This approach facilitates the integration of any new protocol but unfortunately, suffers from performance issues. Our approach is to deploy protocol components in user space as Virtual Network Functions (VNF) inside containers, and to leverage packet processing acceleration toolkits like DPDK (Data Plane Development Kit) [29] to realize high performance close to those of OS kernel.

III. Network Function Virtualization Approach

A *network function* (NF) is a processing function in a network [30]. Traditionally, a network function is deployed on proprietary hardware. Hence, for every customer, a network service provider installs one dedicated hardware per network function (Firewall, DNS, IDS/IPS, etc.). Such network functions deployed on dedicated hardware are called *Physical Network Function* (PNF). Promoted by the ETSI, the *Network Function Virtualization* approach (NFV), as shown in Figure 1, consists in decoupling these network functions from the proprietary hardwares and to implement them as software components in environments providing virtualization capabilities [22]. Such network functions implemented in software are called *Virtual Network Functions* (VNF).

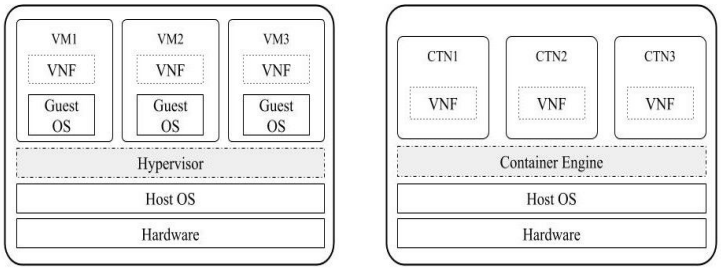

Fig. 2: Hypervisor-based (A) vs. Container-based (B) virtualization

There are two ways to perform virtualization and to package VNFs: *hypervisor-based* virtualization and *container-based* virtualization. The difference between them is shown in Figure 2:

- in the hypervisor-based approach, one virtual machine (VM) with a whole OS is instantiated per VNF resulting in potential overhead, for example, in terms of memory resource consumption;
- in the container-based approach, each VNF running within a container, shares the OS kernel of the host machine with other VNFs.

Because the containerization approach is more flexible, lightweight, portable, scalable, stackable [31] and allows overhead saving, we privileged this latter in our works.

In Figure 2 (B), the role of the *Container Engine* is to automate the deployment of containers, to maintain and update them. There are several container engines, of which the mainstream is *Docker* [31]. Apart from this popular containerization platform, there are others famous platforms like the historical *Linux Containers* (LXC) [32], or more recently *Singularity* [33].

Docker. Although containerization is an old technique introduced in version 2.6 of the Linux kernel, it only became popular with the appearance of Docker. Docker is an open-source platform for developers and sysadmins to develop, deploy, and run applications with containers. By using Docker, we are able to dynamically deploy Transport protocol components as containers inside any device apt to host Docker Engine.

IV. Virtual Transport Protocol

In this section, we describe our approach to deploy and manage Transport components on different entities (computer, server, etc.) within networks. We assume that the considered entities are able to host container engine like Docker. We claim a virtualization-based modular approach where any Transport protocol can be composed from a number of basic Transport functions packaged in virtualization containers. In order to provide unambiguous vocabulary, in the first subsection, we define and illustrate all aspects related to Virtual Transport Protocol concept, among them the need of a control architecture of VTP and a way to drive the composition of protocols. We then introduce a control architecture, called Transport Function Manager (TFM), aiming to manage and deploy protocol components. Finally, we present our approach to compose protocol based on graph.

A. Terminology

We mainly use four terms in our formalism: *Transport Function* (TF), *Virtual Transport Function* (VTF), *Transport Service* (TS) and *Transport protocol*. A *Transport Function*, TF, is a processing unit at Transport level within TCP/IP model. It includes, but not limited to:

- connection management function: ensures opening, maintaining, and closing connection;
- error control function: ensures data integrity;
- flow and congestion control functions: limit sending rate to avoid or react to hosts or routers buffers overflow.

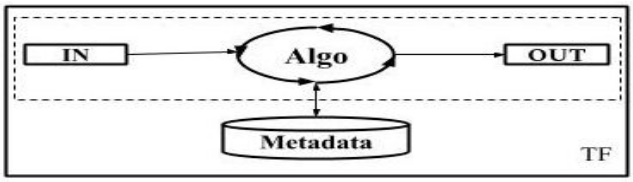

Fig. 3: Transport Function generic model

Figure 3 represents the generic model of a TF. A TF is composed of three mains elements: (1) the IN/OUT data, (2) the algorithm implemented by the TF, and (3) the metadata of the TF. The metadata is essential for the management of the TF and contains information like:

- the *name* or task of the TF; for example “ACK”.
- the *mechanism* implemented by the TF; for example, two TFs performing an ACK task can do it following a cumulative algorithm or a selective mechanism; their mechanisms name will be respectively “cumulative ACK” and “selective ACK”.
- the *dependencies* of the TF; this information indicates which TFs must be executed before the current one; for example, we have to execute *error control function* on segments before executing *reassembly function*.

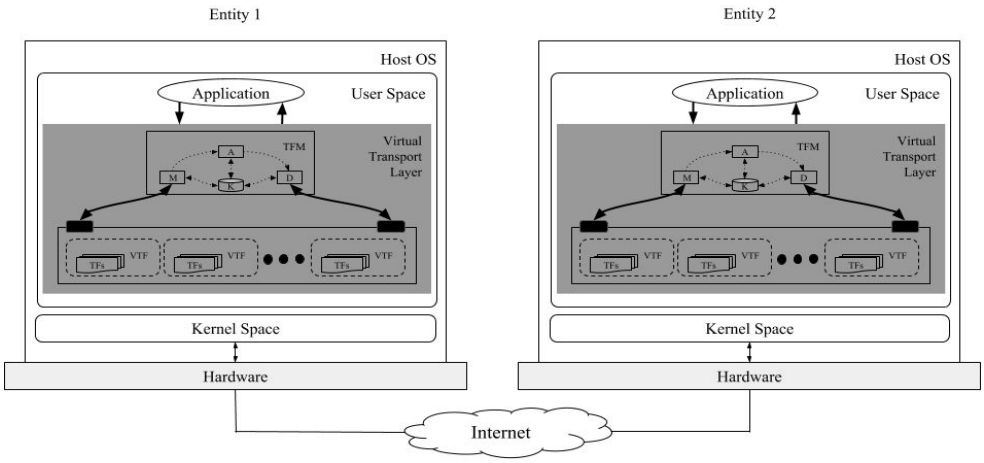

Fig. 4: Overview of TFM and its components

Obviously, the idea of splitting a protocol is not novel and is the so-called modular approach evoked in section 1. In ETP [16], the protocols are composed of so-called micro-protocols or TFs packaged in software components. Such TFs are what we designate by the generic term, *User-space Transport Function* (UTF). As previously mentioned, the code of TFs is most of the time embedded in OS kernel modules; we designate such packaged TFs by the term *Kernel-space Transport Function* (KTF). Instead of having to deal with complex and often unmodifiable OS kernel system, our approach consists in packaging TFs code inside a container and in deploying it on user space of the local entity implied in the communication, or somewhere in the cloud. Such TF, placed inside a container, is called a *Virtual Transport Function* (VTF). In this paper, we focus only on the deployment of VTFs.

A *Transport Service*, TS, is an abstraction of a set of Transport functions. It provides an end-to-end facility to applications. Examples include reliable delivery service, no-loss delivery service, ordered delivery service, partially ordered delivery service, fast delivery service, etc.

A *Transport protocol* is an implementation that provides one or more Transport Services using a specific framing and header format [20], to detail how a Transport sender and a Transport receiver cooperate to provide these services. For instance, TCP protocol is composed by the following services: fully reliable, fully ordered, congestion-controlled and flow-controlled.

B. Transport Function Manager (TFM)

The TFM is a distributed decision system inspired by the MAPE-K cycle [34]. The TFM has to be dynamically deployed in a virtualization container such as Docker container. From service requested by an application, the TFM is able to dynamically build a TS and to deploy all necessary TFs to provide the required service. Figure 4 shows an overview of the TFM and its main components in its current state. We can distinguish *Monitoring*, *Analysis*, *Decision* and *Knowledge Base* components.

Monitoring. The role of *Monitoring* component is to collect events and to reports them in the knowledge base component. Examples of events include new added TF or removed one, OS kernel configuration, started application in user space, etc.

Analysis and Decision. *Analysis* and *Decision* components are together responsible for intercepting applications service requests, interpreting these requests and to dynamically build Transport services to provide requested services. They are also in charge of maintaining and update the Transport Function graph within the Knowledge Base component.

Knowledge Base: Graph-based protocol model. Within a Transport connection, every end-to-end entity has to appropriately handle TFs deployed on it. In order to enable dynamic construction of the required protocol or services, the dependency relationships between the TFs are modeled as a graph. The use of graph facilitates the add, removal, or modification of Transport Function. Furthermore, it is expected to provide an abstraction which will permit to capture the heterogeneity between the TF possible shapes (i.e. TF as a container, VTF; TF as a software component called UTF; or TF as an OS kernel module called KTF). TF graph is a mixed graph G = (V, A, E) where:

- **V** = {TF_1, TF_2, …, TF_n} is the set of TFs deployed on entity;
- **A** is a set of directed links expressing dependency between TFs; an arc (TF_x, TF_y) means that the execution of TF_y requires the execution of TF_x beforehand. For example before execute *Error reporting function,* we have to first detect the error on data through the execution of *Error detection function*.
- **E** is a set of undirected links used when there is not an order in the execution of associated TFs.

A TS is defined in the graph by two paths: one indicates the order of execution of TFs in reception, Rx, and the other indicates the order of execution in transmission, Tx. Figure 5 shows a TF graph where, for example, TS_1 is a ***No-error service*** where: (1) in Rx, TF_3 and TF_0 are executed in this order to detect and report an error, and (2) in Tx, TF_1 is used to recover error by execution of *retransmission function*.

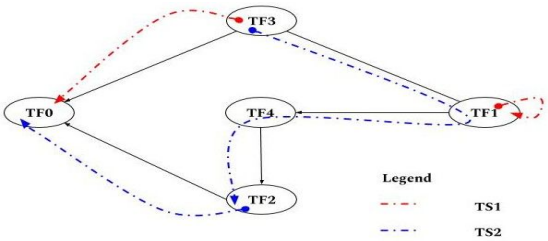

Fig. 5: Example of Transport Function graph

V. Conclusion and Perspectives

In this paper, we propose the *Virtual Transport Protocol* concept consisting of leveraging virtualization principles to permit the dynamically and remotely deployment of Transport components. Based on modular approaches, we propose to divide each *Transport service* (TS) as a set of *Transport Function* (TF) which are packaged in a virtualization container and called *Virtual Transport Function* (VTF). Furthermore, we propose a *Transport Function Manager* (TFM), a control architecture aiming to manage VTFs and dynamically build TS thanks to a proposed TF graph.

Our future work is to build a proof-of-concept of our approach through the implementation of the proposed control architecture. In addition, in this paper, we focus only about the deployment of Transport Function as Virtual Transport Function; we also plane to extend TFM architecture in such a way that it can be able to deploy Transport Functions as OS kernel modules.

References

[1] H. Zimmermann, "OSI Reference Model -- The ISO Model of Architecture for Open Systems Interconnection", Innovations in Internetworking, Artech House, Inc., Norwood, MA, 1988.

[2] R. Braden, "Requirements for Internet Hosts -- Communication Layers", RFC 1122, Internet Engineering Task Force, October 1989.

[3] V. Cerf, Y. Dalal, C. Sunshine, "Specification of Internet Transmission Control Program", RFC 675 (Historic), Internet Engineering Task Force, December 1974.

[4] J. Postel, "User Datagram Protocol", RFC 768, Internet Engineering Task Force, 28 August 1980.

[5] E. Dubois, J. Fasson, C. Donny, E. Chaput, "Enhancing TCP based communications in mobile satellite scenarios: TCP PEPs issues and solutions", Proc. of the 5th ASMS and 11th SPSC, September 2010.

[6] G. P. Fettweis, "The Tactile Internet", IEEE Vehicular Technology Magazine, March 2014.

[7] V. Jacobson, "Congestion Avoidance and Control", Proc. of SIGCOMM' 88, ACM, Stanford, CA, August 1988.

[8] J. Kurose, K. Ross, *Computer Networking - A Top-Down Approach (4th ed.)*. Addison Wesley, 2008.

[9] L. Steven, P. Larry, W. Limin, "Understanding TCP Vegas: Theory and Practice", Technical Report, Princeton University, 2000.

[10] T. Henderson, S. Floyd, A. Gurtov, Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, Internet Engineering Task Force, April 2012.

[11] R. Wang, M. Valla, M. Y. Sanadidi, M. Gerla, "Adaptive Bandwidth Share Estimation in TCP Westwood", Proc. Globecom, 2002.

[12] S. Ha, I. Rhee, L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", ACM SIGOPS Operating System Review, 2008.

[13] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, Internet Engineering Task Force, January 2013.

[14] R. Stewart et al., "Stream Control Transmission Protocol", RFC 2960, Internet Engineering Task Force, October 2000.

[15] E. Kohler, M. Handley, S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, Internet Engineering Task Force, March 2006.

[16] N. V. Wambeke, E. Exposito, C. Chassot, M. Diaz, "ATP: A Micro-protocol Approach to Autonomic Communication", IEEE Transactions on Computers, Vol. 62, No 11, November 2013.

[17] E. Exposito, "Methodology, models and paradigms for a next generation transport layer design", HDR, Institut National Polytechnique de Toulouse, December 2010.

[18] J. Roskind, "QUIC: Multiplexed stream transport over UDP", Google working design document, 2013.

[19] N. Khademi et al., "NEAT: A Platform- and Protocol-Independent Internet Transport API", IEEE Com. Magazine, June 2017.

[20] "Transport Services (TAPS)" https://datatracker.ietf.org/wg/taps/about/, accessed 2018-08-05.

[21] M. Oulmahdi, G. Dugue, and C. Chassot, "Towards a Service-Oriented and Component-Based Transport Layer", International Conference on Smart Communications in Network Technologies (SaCoNeT), 2014.

[22] B. Yi et al., "A comprehensive survey of Network Function Virtualization", Computer Networks 133 (2018) 212-262.

[23] B. A. A. Nunes, M. Mendonca, X.N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present and Future of Programmable Networks", IEEE Communications Surveys and Tutorials, Vol. 16, No 3, 2014, pp. 1617-1634.

[24] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado and S. Shenker, "Extending Networking into the Virtualization Layer", Proc. of HotNets, October 2009.

[25] Z. Niu et al., "Network Stack as a Service in the Cloud", Proc. of the 16th ACM Workshop on Hot Topics in Networks, HotNets-XVI, December 2017.

[26] M. Handley, "Why the Internet only just works", BT Technology Journal, Vol. 24, No 3, July 2006.

[27] M. Tüxen and T. Dreibholz, "The sctplib Userspace SCTP Implementation", 2009, http://www.sctp.de/sctp-download.html, accessed 2018-08-08.

[28] M. Honda et al., "Rekindling network protocol innovation with user-level stacks", ACM SIGCOMM Computer Communications Review, vol. 44, No 2, 2014.

[29] DPDK, https://www.dpdk.org/, accessed 2018-08-12.

[30] "5G", ITU-T Focus Group, IMT-2020 Deliverables, 2017.

[31] Docker, https://www.docker.com, accessed 2018-08-09.

[32] LXC, https://linuxcontainers.org, accessed 2018-08-09.

[33] Singularity, https://www.sylabs.io/, accessed 2018-08-09.

[34] IBM, "An Architectural Blueprint for Autonomic Computing", IBM White Paper 3th Ed., June 2005.