Hindawi
International Journal of Reconfigurable Computing
Volume 2018, Article ID 8785903, 11 pages
https://doi.org/10.1155/2018/8785903

Research Article

Reconfigurable Network Stream Processing on Virtualized FPGA Resources

Qianqiao Chen,[1] **Vaibhawa Mishra**,[2] **Jose Nunez-Yanez**,[1] **and Georgios Zervas**[2]

[1] *University of Bristol, Bristol, UK*
[2] *University College London, London, UK*

Correspondence should be addressed to Qianqiao Chen; qianqiao.chen@bristol.ac.uk

Received 27 September 2017; Accepted 30 January 2018; Published 25 February 2018

Academic Editor: Michael Hübner

The software defined network and network function virtualization are proposed to address the network ossification issue in current Internet infrastructure. Network functions and services are implemented as software applications to increase the programmability of network. However, involving general purpose processors in data plane restricts the bandwidth of network services. Therefore, to keep both the bandwidth and flexibility, a FPGA platform is suggested as a reconfigurable platform to deliver high bandwidth virtual network functions on data plane. In this paper, the FPGA resource has been virtualized by interconnecting partial reconfigurable regions to deliver high bandwidth reconfigurable processing on network streams. With the help of partial reconfiguration technology, network functions on our platform can be configured without affecting other functions on the same FPGA device. The on-chip interconnect system is further evaluated by comparing with existing network-on-chip system. A reconfiguration process is also proposed and demonstrated that it can be performed on our platform. The process can happen in the real time of network services and it is able to keep the original function working during the download of partial bitstream.

1. Introduction

The Internet lays the foundation of our modern world. Computer networks are supporting modern commerce, industry, and the social networks. With the development of computer network, people find it hard to deploy new network technology, services, and protocols in the Internet environment, as the current Internet infrastructure is hard to manage, upgrade, and evolve. In this situation the software defined network (SDN) is suggested to explore the innovation in network design and operation [1].

The SDN proposes to centralize the network control and introduce the ability to program the network. As the network is defined by software applications that is designed and executed in operating systems on general purpose processors, the capital and operational expenditures to manage and upgrade network can be reduced. As complement to SDN, instead of running network applications in real operating system, network function virtualization (NFV) suggests to run these applications in virtual machines [2]. So the hardware resources of network can be isolated and shared.

Although running software network applications in operating system increases the flexibility and programmability of network services, the general purpose processors can limit the bandwidth of these network applications especially on data plane. To maintain both the programmability and bandwidth of network services, FPGA as a reconfigurable device is another outstanding reconfigurable hardware platform for NFV especially in the virtualization of network data plane. To keep the advantage of NFV that share hardware resources to multiple network services, the FPGA hardware platform needs to be virtualized before delivering network functions. However, the virtualization of FPGA resources is not as mature as the virtualization of general purpose processors.

This paper explores the virtualization technology of FPGA resources to deliver high bandwidth virtual network functions. A NFV platform is implemented in this paper to deliver network data plane stream processing on virtualized

FPGA resources. The contribution of this paper includes the following:

(i) An architecture is implemented to virtualize FPGA resource. In the proposed architecture, the technology of partial reconfiguration and on-chip interconnect is adopted to isolate FPGA resource. The FPGA resource can also be directly attached to the 10 G Ethernet without involving processing unit in the network data plane.

(ii) The on-chip interconnect system in the proposed architecture is evaluated and compared with existing interconnect system.

(iii) A reconfiguration process that is able to switch partial reconfigurable network functions in real time and in a packet loss free manner is proposed and demonstrated.

The rest of the paper is organized as follows: Section 2 introduces the background. The overview architecture of the proposed platform for reconfigurable network stream processing is explained in Section 3. The on-chip interconnect in the proposed system is evaluated in Section 4. The proposed reconfiguration process is compared with typical partial reconfiguration process in Section 5. Section 6 demonstrates the experiment to validate the platform and demonstrate the proposed reconfiguration process. Section 7 concludes this paper.

2. Background

The SDN is a network paradigm that aims to decouple the control plane and data plane of the network. The SDN enables centralized control and management on the distributed data plane of a network [3]. The SDN also suggests a standard software control plane which can be independent from its hardware [4]. Although the SDN only suggests a software control plane, software data plane is often adopted as a complement of the SDN concept. Especially, in data centers, software network switch is often used between virtual machines that is deployed on the same server [5].

These software applications on network control and data plane enlighten the concept of NFV. Network functions are no longer treated as functions of an ASIC based hardware platform built with specific purpose. Instead, they are now software applications running on the operating system of general purpose processors. Therefore, the virtualization technology in computer system can be adopted to isolate and share network resource to multiple tenants [6]. The NFV first proposed to virtualize the control plane by using the SDN technology. Then both the control and data plane of a network function are virtualized to deliver various network services such as firewalls and load balancers. Furthermore, services beyond the OSI application layer (such as video processing services) are suggested as a complement of NFV especially in data center environment [7]. The NFV inherent many features and advantages from the SDN. For example, network services can be fast and easily deployed by using NFV technology [8]. The network can be adjusted based on real-time traffic and the requirement of the network services. But the NFV also has its unique advantages such as sharing the physical network resources to multiple tenants [6].

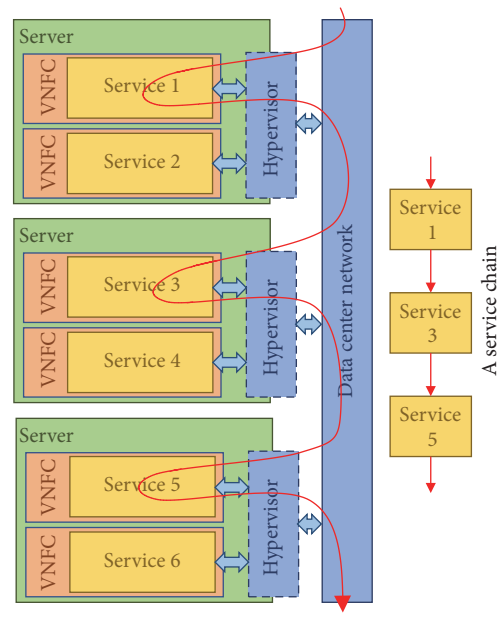

Figure 1: A typical infrastructure of NFV and the service chain.

Figure 1 shows a typical infrastructure of NFV in data centers. Servers are the common hardware platform of NFV in data centers [9]. The software that enables the virtualization of servers is called hypervisor (also known as virtual machine monitor) [10]. The hypervisor is responsible to isolate and allocate physical resources on the computing platform to individual virtual machines (also known as virtual network function container in the terminology of NFV). The hypervisor in NFV is often specialized for network virtualization. For example, the hypervisor is often equipped with a hardware or software network switch. Services deployed in virtual machines can be connected on the demand of users to build a chain of services as marked in the red line of Figure 1 [11].

Although moving network functions from hardware to software decouple functions from its physical platform, it reduces the performance of network functions. Therefore, additional hardware accelerators are often required in the network to meet the performance requirement. Moving back to ASIC accelerators increases the coupling degree of control and data planes which go against the design philosophy of NFV. So FPGA as a reconfigurable but high performance platform is adopted in this paper [12].

Similar to the NFV on general purpose processors, NFV on FPGA should also be built on the virtualization technology of FPGAs to share physical resources to multiple tenants. Partial reconfiguration has become the key technology to enable the virtualization of FPGA resource. Reference [13] explores the integration of partial bitstream download process with the operating system. The hardware environment of the FPGA virtualization is implemented in [14], especially, PCIe interface has been set up as the interface between processing unit and programmable logic. Reference

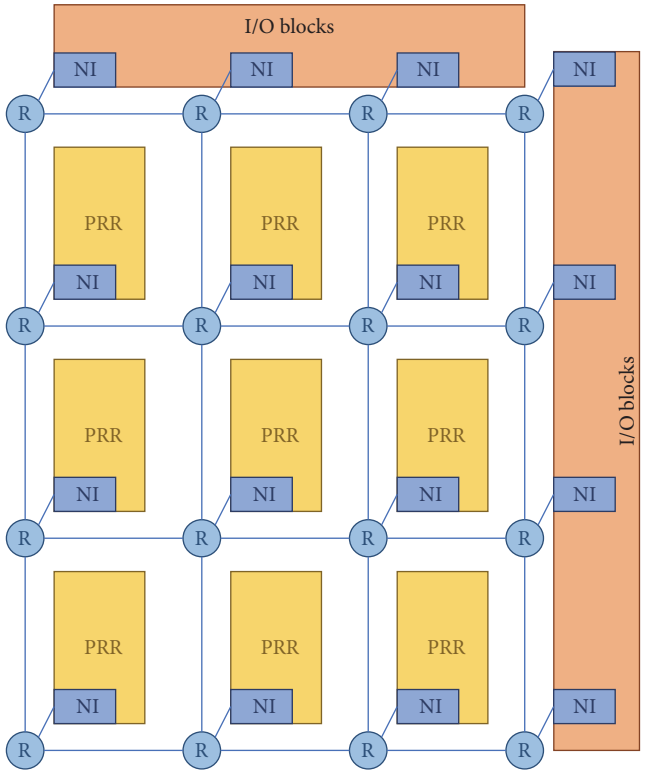

Figure 2: A typical NoC based architecture for FPGA virtualization.

[15] improves the virtualization by enhancing the software support especially for the data center environment.

Both the data and control plane are coordinate by the processing unit in above researches. However, to virtualize FPGA for network functions, involving general purpose processors in data plane can reduce the performance especially the bandwidth. Therefore, instead of using processing unit, FPGA based interconnect architectures are taken into consideration in the virtualization of FPGA resources. The research in [16] proposes a network-on-chip based design to share FPGA resources. The research in [17] extends the NoC based FPGA virtualization in the cloud computing. A typical NoC based FPGA virtualization architecture is shown in Figure 2. It is often composed of several partial reconfigurable regions where accelerators can be deployed. Data communication between the deployed accelerators are transferred through an interconnect system. I/Os are often attached to the interconnect as well to communicate with off-chip network. In this paper, following the concept of FPGA virtualization, a FPGA virtualization platform is implemented to deliver reconfigurable high bandwidth network stream processing on the data plane of NFV.

3. Overview Architecture

In this paper, the aim of FPGA virtualization is to support NFV. In other words, the physical FPGA platform should be shared by multiple network services requested by many users. In addition, the adoption of FPGA is to virtualize the network

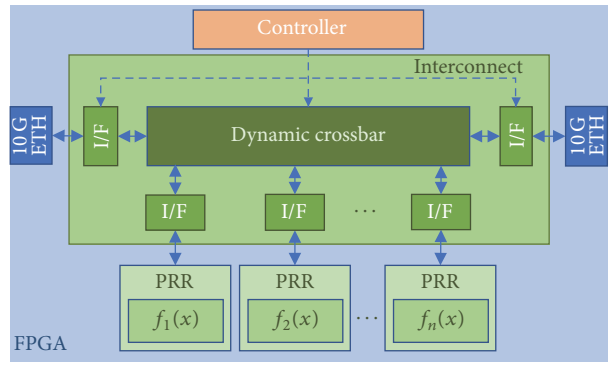

Figure 3: The overview of the proposed FPGA virtualization architecture for NFV platform.

function on data plane. So the virtualized network functions should support high bandwidth. Therefore, the virtualized FPGA platform should support the following features: (a) network functions should be configured independently, (b) network functions should have high performance interfaces, and (c) network functions should share the network interfaces of FPGA board.

The typical configuration of FPGA needs to reprogram entire FPGA device. However, when a FPGA is shared by multiple network functions, reprogramming one of the functions needs to turn off all the other functions on the board, which affect functions belonging to other services and users. Therefore, the partial reconfiguration technology is adopted. It partitions the FPGA into one static region and several dynamic partial reconfigurable regions. The program of partial reconfigurable regions will not affect the function in static region and other partial reconfigurable regions.

The design feature of partial reconfiguration requires static interface between dynamic regions and the static region. So all the deployed network functions should be attached with a standard interface to enable the partial reconfiguration technology. The standard AXI4-stream is utilized as the interface in this paper. To support high bandwidth network function, an on-chip interconnect system is also implemented in the static region to establish data communication between dynamic regions.

FPGA devices often include limited number of network interfaces. So the network interfaces should also be shared by network functions when a FPGA platform is built to support NFV. Therefore, the 10 Gbps Ethernet interface is attached to the interconnect and makes the interconnect forward data at the Ethernet packet level. So all the dynamic regions are able to have access to the off-chip network through the interconnect.

The structure of the proposed FPGA virtualization for NFV is suggested in [18] and is shown in Figure 3. It includes a number of partial reconfigurable regions (PRR) and an interconnect system. The 10 Gbps Ethernet systems are also attached to the interconnect system to transfer data with off-chip network directly. A controller is also included to control the interconnect system and orchestrate the reconfiguration process of virtual functions (introduced in Section 5). All the

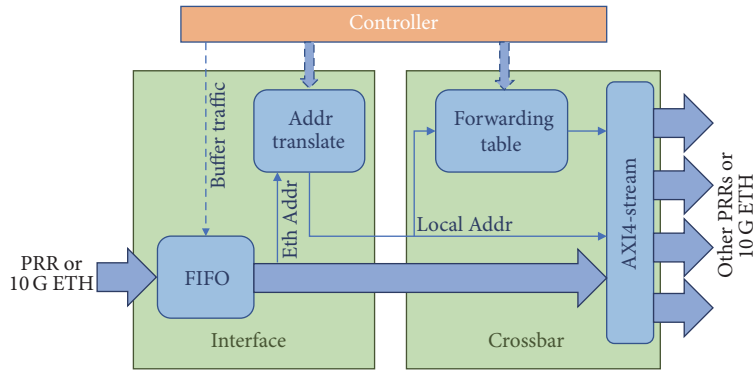

Figure 4: The detailed structure of the interface and dynamic crossbar of the interconnect.

modules are running at 156.25 MHz, which is the frequency of the 10 G Ethernet system with 64-bit data width.

A dynamic interconnect system is implemented and forwards data between PRRs in the proposed platform. It is composed of a dynamic crossbar and interfaces to PRRs. The data are forwarded based on the local address signal in parallel with the data signal (as the USER channel in AXI4-stream standard). Each PRR is identified by a unique address. The local destination address of a packet is added in the interface. And the crossbar forwards packets according to the attached local address. The architecture of the crossbar and interface will be further introduced in next paragraph. The 10 Gbps network ports are also connected with the interconnect system. All PRRs are decoupled from physical interfaces and are directly attached to interconnect system as pluggable independently accessible regions. This enables reconfigurable processing on network streams. The whole FPGA virtualization system is managed from the central controller. Update information is transferred through the central controller to set the value of memory-mapped control registers and change the port map of the dynamic crossbar. The central controller is also responsible to perform the function reconfiguration process. The detailed information of the process is given in Section 5.

The implementation detail of the dynamic crossbar and interface is shown in Figure 4. The interface locates at the edge of the interconnect system to bridge PRRs and PHYs with the crossbar. The crossbar and interfaces are connected following the AXI4-stream handshake standard.

The Ethernet address is translated into local address according to the table in the interface. This table can be updated from the controller through AXI4-lite interface to enable real-time control on the interconnect system. The local address is transferred in parallel with the data (as the USER channel of AXI4-stream) until reaching its required PRR. The interface is also responsible to buffer or release the traffic. FIFO is added to temporarily store the data during the reconfiguration of the interconnect. The FIFO can be controlled from the central controller.

A forwarding table storing the map information between local address and output port is stored in dynamic crossbar. When a packet is received by the crossbar, its local address (attached by the interface) is checked and the packet is forwarded to the related output port by looking up the forwarding table. The forwarding table is dynamic and can be updated from the central controller through an AXI4-lite interface.

Since all network functions are only attached to the interconnect system and are clearly decoupled from the high speed I/Os, the flexible FPGA virtualization platform can deploy and involve network functions where and when required. Network stream processing unit can be reconfigured on PRRs on per port, per independent flow, or per virtual network basis. PRRs can also be reconfigured into requested functions at run time without service disruption of or loss of data (introduced in Section 5).

Therefore, a reconfigurable NFV platform with the following features is established: (a) independent virtual FPGA resource or regions; (b) shared network interfaces by network functions; (c) network developers being able to enhance the already deployed functions in real time and in packet loss free manner.

4. Evaluation of the Interconnect

The interconnect system plays a significant role in the FPGA virtualization system, as all the received traffic needs to go through the interconnect system before reaching its destination. A low performance interconnect structure will limit the performance of all the deployed network functions. Therefore, the interconnect system of the proposed FPGA virtualization platform is evaluated.

To put the resource utilization and network performance of our interconnect system into perspective, the interconnect system is compared with the public available CONNECT [19, 20] and SOTA [21] network on chip. The SOTA mainly targets the ASIC design. When mapping FPGA platform, the SOTA needs large number of resources. The CONNECT tries to solve this problem at the cost of reducing maximum router frequency [22, 23]. In the following presentation, the resource utilization in synthesis result of all the interconnect system

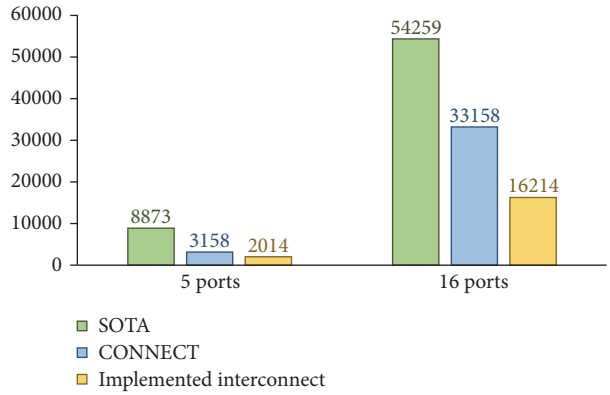

Figure 5: The LUT resource utilization of CONNECT, SOTA, and the interconnect system implemented in this paper.

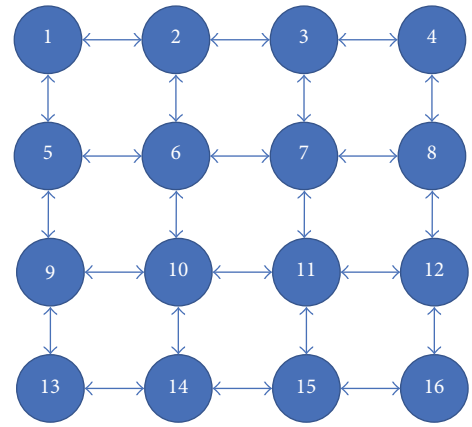

Figure 6: The pattern for the static part of the hybrid traffic.

will be compared. Then the cycle accurate simulation result will be collected to evaluate the network performance.

Routers with 5 and 16 ports of all the three interconnect systems are set up, respectively. The data width of both CONNECT and our interconnect system is 64 bits. The SOTA only supports 32-bit data width. And the resource utilization in terms of LUTs is shown in the bar graph in Figure 5. The implemented interconnect system uses the least resource among all the three interconnect systems.

To compare the network performance, two traffic patterns are set up and the load-delay curves are measured. The first traffic pattern is uniform random traffic, where the destination of each packet is set randomly. The second traffic pattern is a hybrid traffic of two patterns. 90% of the hybrid traffic is a static traffic pattern. The map of input and output ports of the generated flows is fixed as shown in Figure 6. And the loads of receive ports are equal to each other. 10% of the hybrid traffic is uniform random traffic, which is the same as the first traffic. The second traffic depicts a traffic pattern where deployed functions heavily transfer data with several other functions in the system and also talk to the rest functions occasionally.

The result is shown in Figure 7. When all the three interconnect systems are running at the same frequency, our interconnect system can support maximum 4.3 Gbps under uniform random traffic and 5.8 at hybrid traffic, followed by CONNECT with data width of 64 bits as shown in Figures 7(a) and 7(b). However, the higher maximum bandwidth is at the cost of delay. Our interconnect system needs additional 40 ns at the situation of random traffic and 20–35 ns under hybrid traffic. The three interconnect systems are compared at their design frequency. Under both traffic patterns, the delay of our interconnect system is similar to the 64-bit CONNECT. But our interconnect can is still able to support higher bandwidth as shown in Figures 7(c) and 7(d).

The SOTA system has the lowest performance under all the circumstances. This is because SOTA only supports 32-bit data width, which limits the performance extremely. Different from our interconnect system and CONNECT, in the SOTA system, the control information including address is transferred in band with the DATA. This also introduces overheads. The CONNECT system has lower latency, especially when the system is not heavily loaded. This is because CONNECT is designed for a computing system, where delay is more important than bandwidth. Our system can support higher bandwidth which is important in a network system. Apart from bandwidth, 10 G Ethernet interface and Ethernet address translation are also attached to make our system work more directly with off-chip network.

5. Reconfiguration Process

The download process of partial bitstream takes time when performing partial reconfiguration. As the PRR cannot perform any function during this period, the adoption of partial reconfiguration can introduce loss of service. To avoid service loss, a function reconfiguration process is proposed which can be free of packet loss on our NFV platform. This reconfiguration process is introduced in this section.

Instead of occupying all PRRs by network functions, a backup (empty) PRR is reserved only for the reconfiguration process. As the empty backup PRR does not receive and send any traffic, the bitstream for this backup PRR can be downloaded without disturbing the existing traffic in the system. When a function reconfiguration (i.e., from IP parser to UDP parser in this example) is requested, partial bitstream of the requested function will be firstly loaded on the backup PRR. During the deployment of the partial bitstream on the backup PRR, the rest of the platform including all the active PRRs will not be affected or interrupted. The backup PRR with the requested function will not be set into use until the reconfiguration is finished. When the deployment of partial bit stream on the backup PRR is finished (the requested function is ready), the port map of the interconnect system will be updated to forward the related data flows to the backup PRR and stop forwarding data to the previous active PRR. The previous backup PRR becomes active and the previous active PRR takes the role of the new backup PRR for future function reconfiguration.

As shown in Figure 8, the PRRs, interconnect system, and the interfaces should be orchestrated properly during the reconfiguration process. The central controller takes the

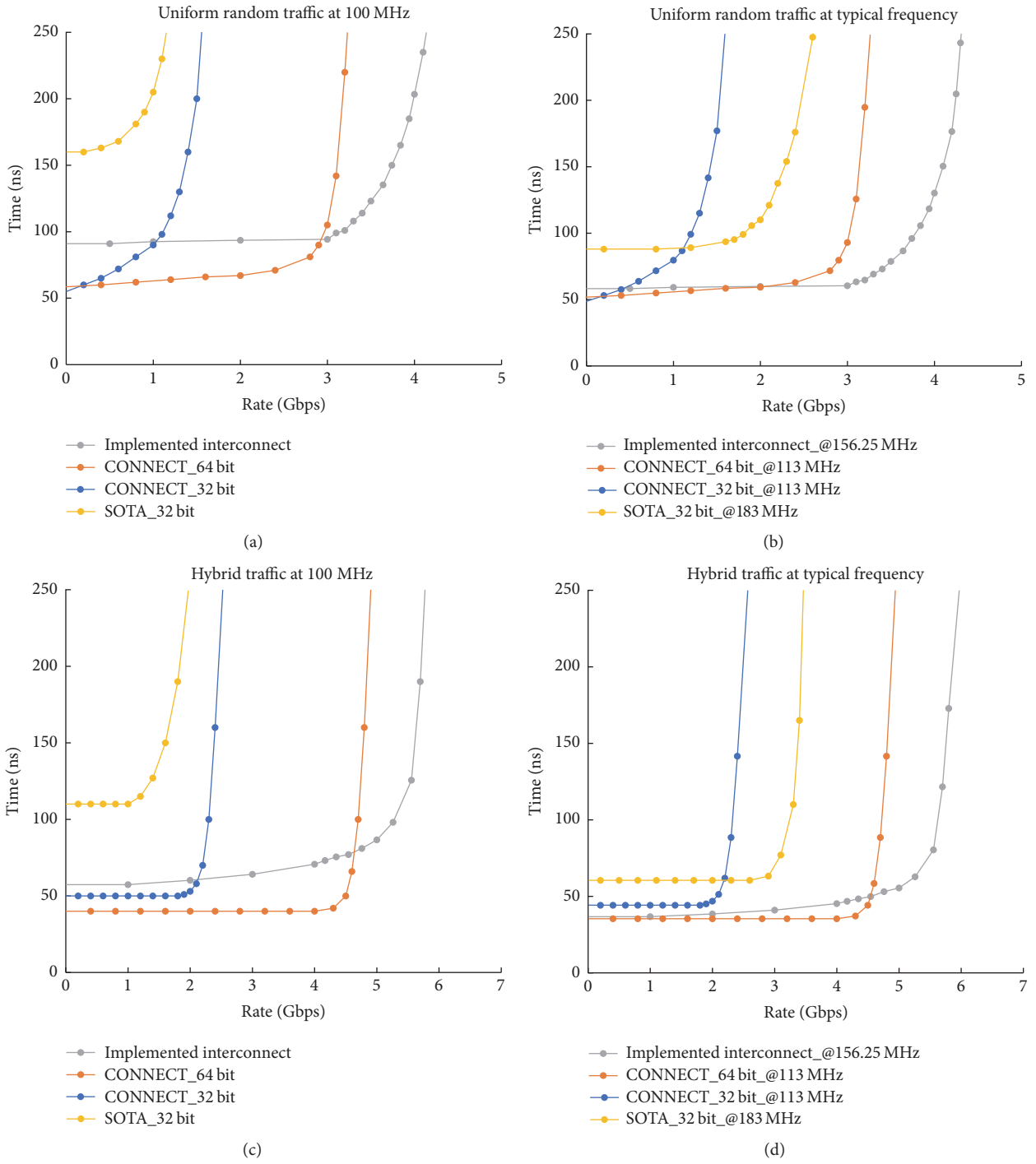

Figure 7: The comparison between the implemented interconnect system and the existing SOTA and CONNECT interconnect.

responsibility of orchestrating the reconfiguration process. Five steps need to be performed by the controller. Partial bit stream of the requested function is downloaded from the host PC to the backup PRR in the first step (a). The traffic needs to be buffered in the second step. (b) The controller then updates the port map of the interconnect in step (c). The traffic will then be released from the interface and will be forwarded through the PRR equipped with the new requested function in step (d).

The proposed reconfiguration process can increase the performance of the reconfiguration, since the platform is still functioning during the download of partial bit stream. The traffic only needs to be buffered when the port map of the interconnect is being updated. The buffer time to enable

Table 1: Resource utilization and partial bit file size.

VNF processor	Flip flop	LUT	Partial bit file size (KB)	Download time (us)
Ethernet parser	493	290	714	1785
IP parser	1268	1001	792	1980
UDP parser	2072	1451	846	2115

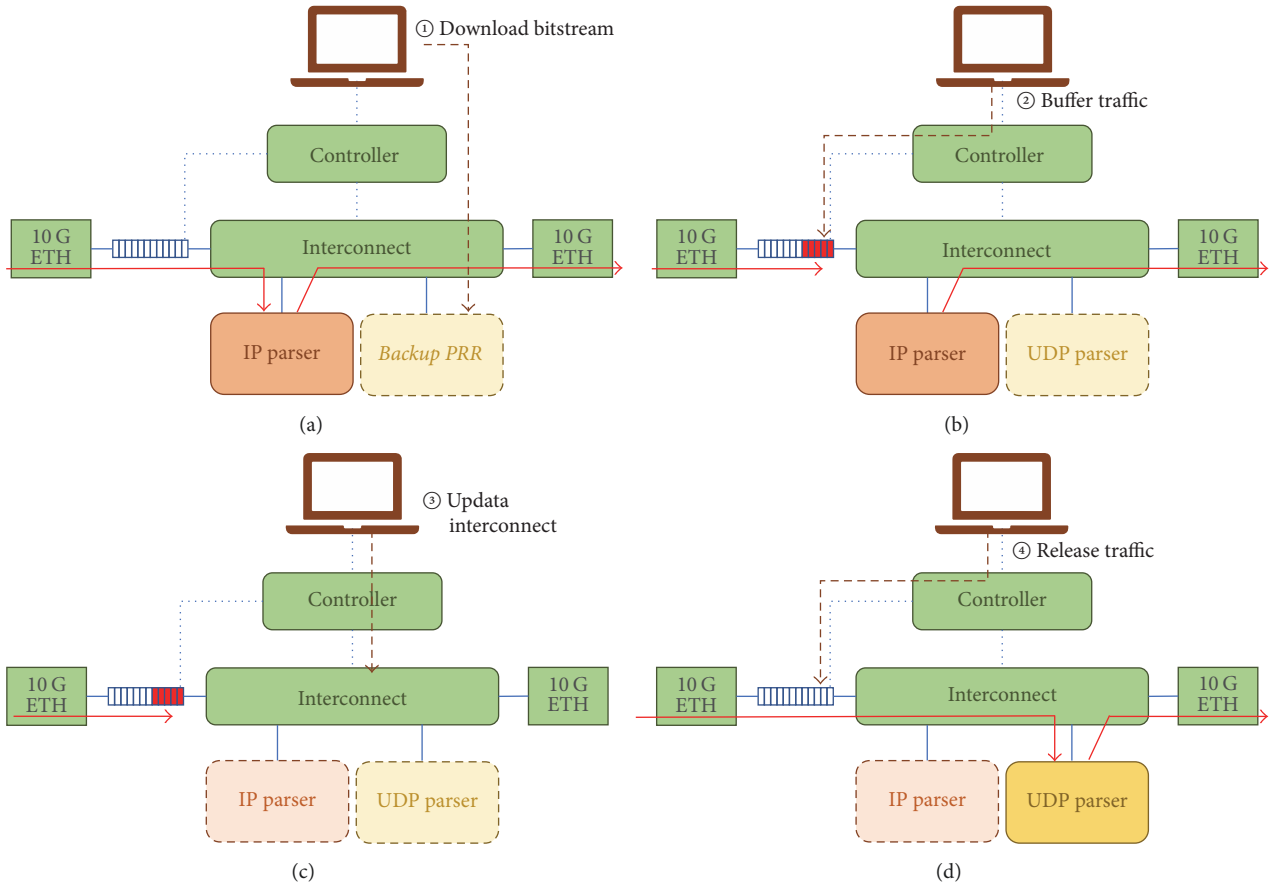

Figure 8: The proposed reconfiguration process that supports function reconfiguration in real time has the following four steps: (a) downloading partial bitstream to the backup PRR; (b) buffering the traffic; (c) updating the interconnect; (d) releasing the traffic.

packet loss free for the common reconfiguration process and the proposed process is compared in the following paragraphs.

In the common process of partial reconfiguration, the traffic needs to be buffered during the download of the bit stream to enable packet loss free reconfigurable services, since the PRR is not functioning during the period of reconfiguration. The required download time of the partial bit file according to their sizes are shown in Figure 9. The partial reconfiguration is requested through Vivado using the typical configuration of the reconfiguration controller (ICAPE2 at 100 MHz). As the partial bit file needs to be sent to the reconfiguration controller, the size of the bit file has a strong influence on the time of reconfiguration. The resource utilization, size of partial bit file, and download time of a set of 64-bit network functions are shown in Table 1. The Ethernet parser which has minimum partial bit file size needs 1785 microseconds to be downloaded. The UDP parser takes the maximum download time which is 2115 microseconds. Following the common reconfiguration process, these times translate to excessive buffering resources as shown in Figure 9.

During our proposed reconfiguration process, data flows need to be buffered only when the interconnect system is being updated, since the previous function is still running in the active PRR. The buffer time required in this process is only the update time of the interconnect system. And the size of the partial bit stream will not affect the buffer time. As introduced in Section 3, the port map of interconnect system is updated through AXI4-lite interface. The size of the update data will influence the buffer time. The size of each destination-output port pair of the forwarding table is 8 bytes. The time and buffer needed to reconfigure the interconnect system in packet loss free manner are shown in Figure 10.

Table 2: Resource utilization.

Component	Flip flop	LUT	BRAM
NoC interface	446	267	3
NoC router	10398	22268	0
Central controller	994	432	2

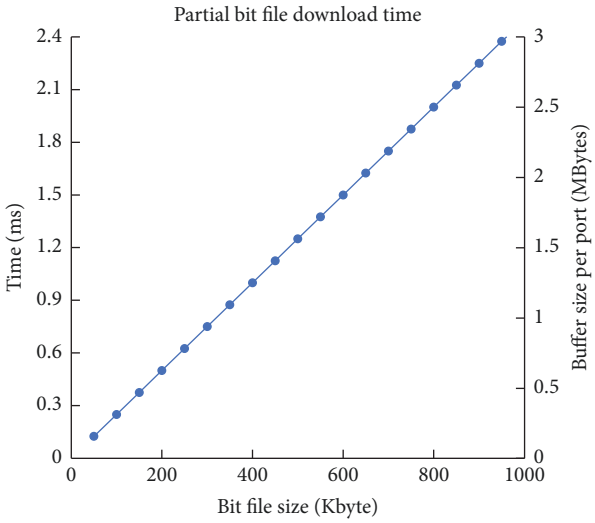

Figure 9: Time and buffer needed for packet loss free partial bitfile download.

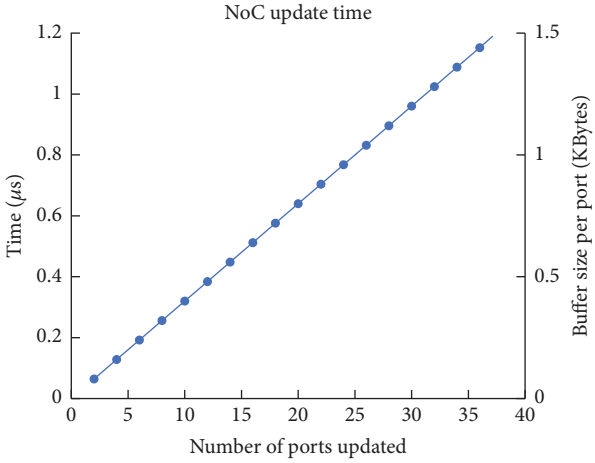

Figure 10: Time and buffer needed for packet loss free NoC update time.

The result indicates that the time to update interconnect system is at the level of microsecond (Figure 10) compared to the common partial bit stream download time which is at the level of hundreds of microseconds or even milliseconds (Figure 9). As the proposed NFV platform is still able to process data flows during the download time of the partial bit stream, the proposed reconfiguration process can shorten the time required to buffer the traffic. Therefore, the proposed process enables a packet loss free reconfiguration between network functions where no data or packets are lost during the process. By adopting the proposed process, network developers will be able to change network functions on demand without stopping the traffic.

6. Experiment and Demonstration

Table 2 shows the resource requirement for an interconnect system with 16 ports. The NFV platform is implemented on the NetFPGA SUME board with Xilinx Virtex-7 690T FPGA chip [24]. The Network Master Pro MT1000A from Anritsu is used to generate and analyze the proposed NFV platform. The accuracy of the latency of the traffic analyzer is 100 nanoseconds. The traffic analyzer also has a built-in 100-nanosecond latency (measured in a loopback mode). This built-in latency has been removed in the following result. The network analyzer is connected with the 10 Gbps SFP+ optical transceivers on the NetFPGA through one-meter single mode fiber. As the accuracy of the traffic analyzer is 100 nanoseconds, the latency of the fiber (5 nanoseconds per meter) is ignored.

To demonstrate the interconnect system on real hardware, an Ethernet switch is set up. The interconnect system firstly runs at the frequency of the 10 G Ethernet system (156.25 MHz). Then the interconnect system also runs at higher frequency (200 MHz). As shown in Figure 11(a), in the system at 156.2 MHz, the 10 G Ethernet system is directly connected with our interconnect system. And in the system at 200 MHz (Figure 11(c)), the clock converters are inserted between the 10 G Ethernet system and the interconnect system. The traffic on both configurations is the same: receiving two data flows and forwarding to the same output 10 G Ethernet interface.

The influence of throughput on the latency is shown in Figure 11(c). And the influence of packet size is shown in Figure 11(d). The 156.25 MHz system can achieve ultralow latency (500 nanoseconds and 1 microsecond for minimum and maximum throughput, resp.), for the 200 MHz system, 1.2 microseconds and 1.9 microseconds for minimum and maximum throughput, respectively. The clock converter in 200 MHz system introduces more delays, because the clock converter needs to insert invalid clock cycles (low tvalid clock cycles in AXI4-stream) in a packet when it converts from a clock domain with higher frequency (200 MHz) to a clock domain with lower frequency (156.25 MHz). But the transmit side of 10 G Ethernet system always needs a complete packet (no invalid clock cycles in the middle of a packet) from the inner system of FPGA. So there must be a FIFO to store a

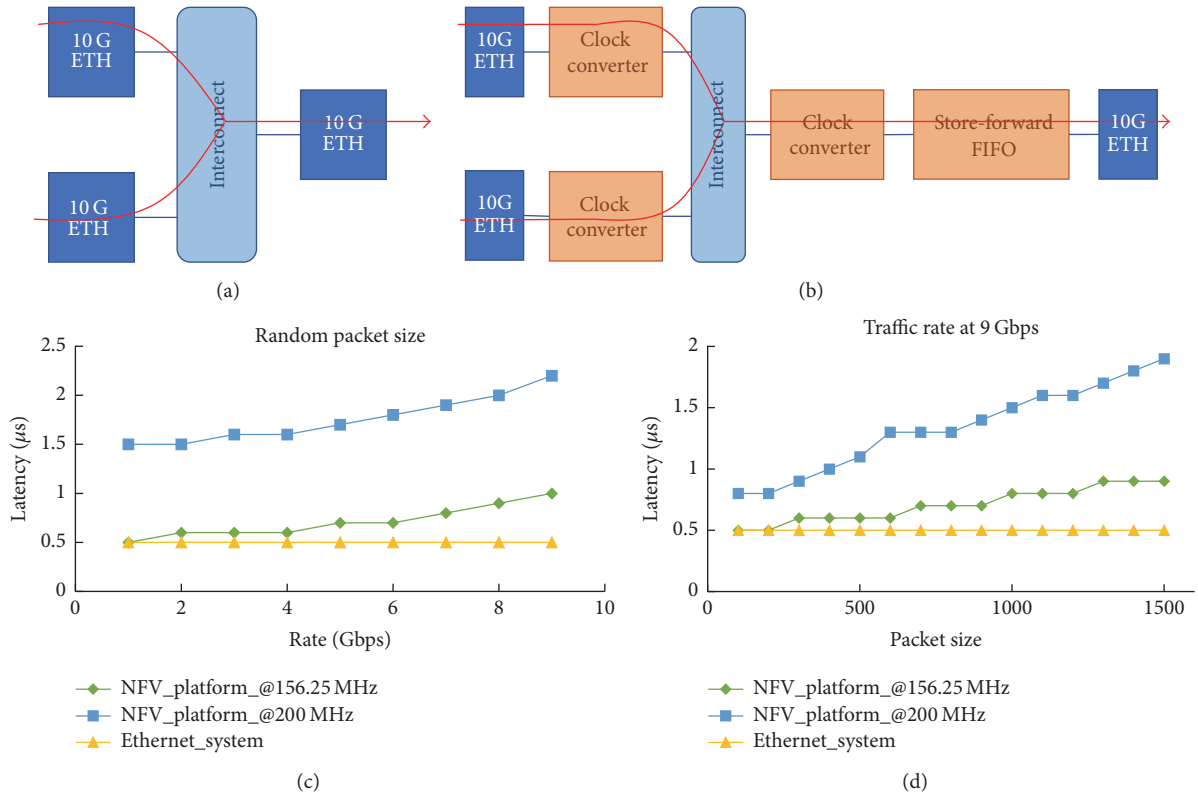

Figure 11: The experiment was set up at 156.25 MHz (a) and 200 MHz (b). And the latency under different data rate (c) and Ethernet packet size (d).

complete packet, eliminate invalid clock cycles introduced by the clock converter, and then send the entire packet to the 10 G Ethernet system. This store-forward FIFO increases the latency. Therefore, the NFV platform runs at the frequency of 10 G Ethernet system which is 156.25 MHz.

A reconfiguration between IP and UDP parser is also demonstrated. The demonstrated process is shown in Figure 12. IP parser is initially deployed in an active PRR and is processing the current data flow. Then the bit stream for the UDP parser is downloaded in the backup PRR. After the proposed reconfiguration process, the traffic is forwarded to the backup PRR to make the deployed UDP parser process the traffic. Then the previous active PRR into UDP parser is configured and the traffic switches back to the active PRR by using the proposed reconfiguration process.

The number of parsed packets in each PRR is recorded as shown in Figure 13. The backup PRR starts parsing the traffic in the 20th to 30th second. The packets are processed in the backup PRR from 30th to 70th second. The previous active PRR is then configured into UDP parser and is then put into use after 70 seconds. The backup PRR becomes available again for reconfigurations in the future. The total number of packets is recorded by the traffic analyzer which is 77518546. The count of packets parsed in each PRR is 42708766 (original active PRR) and 34809780 (original backup PRR). No packet is lost during the whole process as the total number of counted packets in FPGA is the same as the total number of transmit packets recorded by traffic analyzer (34809780 + 42708766 = 77518546).

7. Conclusion

To keep both the bandwidth and the flexibility of network services, this paper suggests FPGA as the hardware platform of NFV to deliver network data plane functions. To isolate and share FPGA resource to multiple network services, this paper proposed a NFV platform to deliver reconfigurable network data plane stream processing based on the virtualization of FPGA resource.

A FPGA virtualization architecture is implemented. It includes several partial reconfigurable regions communicating through on-chip interconnect system. The performance of our on-chip interconnect system is compared with existing network-on-chip architecture. The result shows that our interconnect can support higher bandwidth up to 4.3 Gbps under random traffic at 156.25 MHz. A reconfiguration process to switch partial reconfigurable network functions in real time is proposed and demonstrated. In this process, network functions can keep running during the download of partial bitstream. Experiment has been done to evaluate the proposed platform in real traffic. In the experiment, the platform is forwarding network stream from two input ports to one output port. It is possible to run at maximum 9 Gbps with the latency of 1 microseconds.

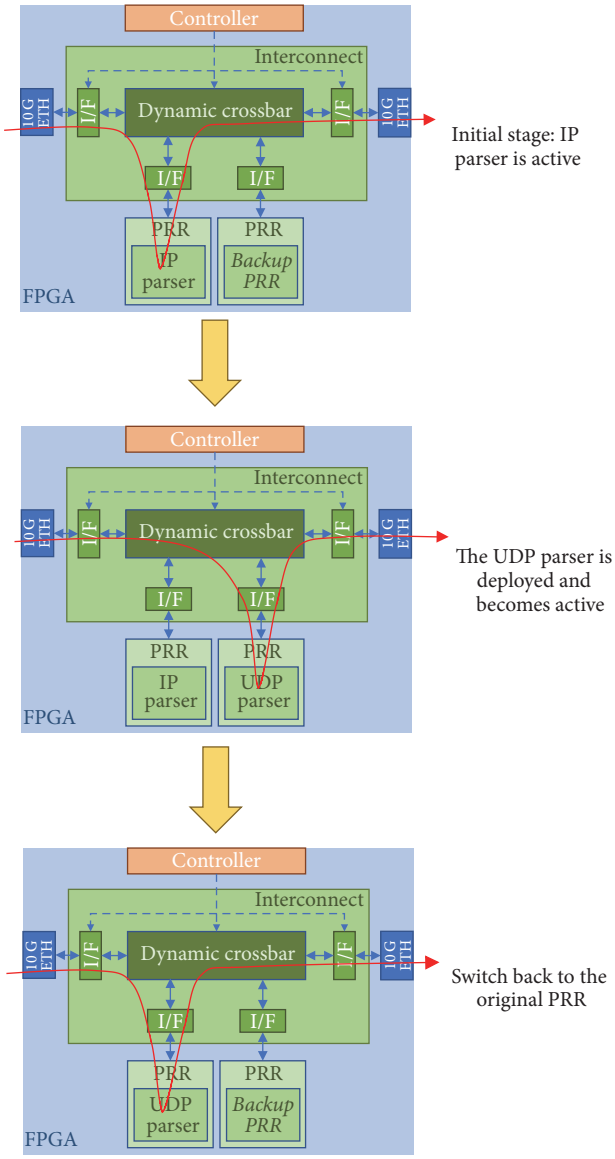

Figure 12: The demonstrated reconfiguration process.

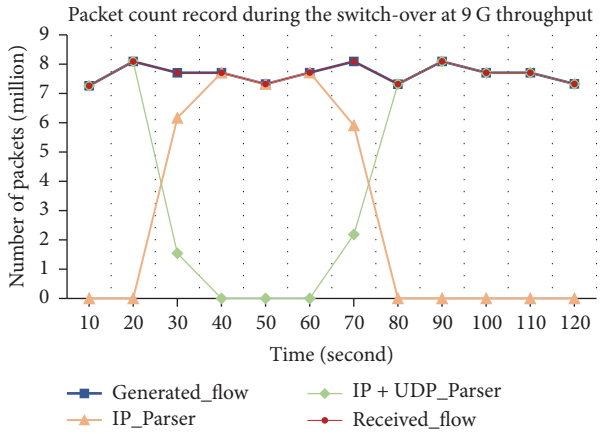

Figure 13: Network function virtualization in data centers.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the EC H2020 dReDBox Project with Grant Agreement no. 687632.

References

[1] "Software-Defined Networking (SDN) Definition," Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/sdn-definition/. [Accessed: 26-Sep-2017].

[2] "Network Function Virtualization," ETSI. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv. [Accessed: 26-Sep-2017].

[3] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[5] B. Pfaff et al., "The Design and Implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, pp. 117–130, CA, USA, 2015.

[6] "Network Functions Virtualisation White Paper 1," ETSI. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf. [Accessed: 13-Sep-2017].

[7] H. Koumaras, C. Sakkas, M. A. Kourtis, C. Xilouris, V. Koumaras, and G. Gardikis, "Enabling agile video transcoding over SDN/NFV-enabled networks," in *Proceedings of the International Conference on Telecommunications and Multimedia, TEMU*, pp. 1–5, Greece, July 2016.

[8] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: state-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[9] Y. Li and M. Chen, "Software-defined network function virtualization: a survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[10] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 655–685, 2016.

[11] "Network Functions Virtualisation White Paper 3," ETSI. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf. [Accessed: 13-Sep-2017].

[12] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, "Reconfigurable network systems and software-defined networking," *Proceedings of the IEEE*, vol. 103, no. 7, pp. 1102–1124, 2015.

[13] C. H. Huang and P. A. Hsiung, "Hardware resource virtualization for dynamically partially reconfigurable systems," *IEEE Embedded Systems Letters*, vol. 1, no. 1, pp. 19–23, 2009.

[14] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA accelerators for efficient cloud computing," in *Proceedings of the IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom '15)*, pp. 430–435, Vancouver, Canada, November 2015.

[15] O. Knodel, P. Lehmann, and R. G. Spallek, "RC3E: Reconfigurable accelerators in data centres and their provision by adapted service models," in *Proceedings of the 9th International Conference on Cloud Computing, CLOUD 2016*, pp. 19–26, USA, July 2016.

[16] J. Yang, L. Yan, L. Ju, Y. Wen, S. Zhang, and T. Chen, "Homogeneous NoC-based FPGA: The foundation for virtual FPGA," in *The Foundation for Virtual FPGA," in 10th IEEE International Conference on Computer and Information Technology*, pp. 62–67, UK, July 2010.

[17] H. L. Kidane, E.-B. Bourennane, and G. Ochoa-Ruiz, "NoC Based Virtualized Accelerators for Cloud Computing," in *Proceedings of the 10th IEEE International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2016*, pp. 133–137, France, September 2016.

[18] Q. Chen, V. Mishra, and G. Zervas, "Reconfigurable computing for network function virtualization: A protocol independent switch," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs, ReConFig 2016*, Mexico, December 2016.

[19] M. K. Papamichael and J. C. Hoe, "CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '12)*, pp. 37–46, ACM, February 2012.

[20] M. Papamichael, "CONNECT: CONfigurable NEtwork Creation Tool." [Online]. Available: http://users.ece.cmu.edu/~mpapamic/connect/. [Accessed: 26-Sep-2017].

[21] Stanford Concurrent VLSI Architecture Group, "Enabling Technology for On-Chip Networks." [Online]. Available: http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/WikiStart. [Accessed: 26-Sep-2017].

[22] A. Monemi, C. Y. Ooi, and M. N. Marsono, "Low latency Network-on-Chip router microarchitecture using request masking technique," *International Journal of Reconfigurable Computing*, vol. 2015, Article ID 570836, 13 pages, 2015.

[23] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, vol. 54, pp. 60–74, 2017.

[24] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, article no. 61, pp. 32–41, 2014.