REoN: A Protocol for Reliable Software-Defined FPGA Partial Reconfiguration over Network

Vaibhawa Mishra, Qiaonqiao Chen, Georgious Zervas
Department of Electrical and Electronics Engineering
University of Bristol
Bristol, United Kingdom
Email: vaibhawa.mishra@bristol.ac.uk

Abstract— **This paper presents and defines a Reconfiguration over Network (REoN) protocol. It is a solution for a FPGA-based dynamically reconfigurable system, that offers partial (re)programming over the network without the need of a local/embedded soft/hard processor. This protocol can transport partial bit files from centralized control and management system via network resource management API to a FPGA empowered network node, using standard 10 Gbps Ethernet. This work architects and introduces a proprietary lightweight connection oriented protocol stack, which guarantees reliability over standard UDP/IP protocol. Hardware stack for standard networking protocols including remote reconfiguration engine directly interfaced with Xilinx Internal Configuration Access Port (ICAP). This minimizes FPGA resource requirements in re-programming the FPGA. The presented work is an enabling technology for a range of applications such as reconfigurable computing enabled Network Function Virtualization (NFV), function disaggregation on data centres empowered by FPGA/SoCs, as well as Internet of Things (IoT).**

Keywords—Partial Reconfiguration, network protocols, remote dynamic reconfiguration

I. Introduction

FPGAs are being used in application specific processing such as video, image or general purpose computing for more than twenty years. In recent years, FPGAs have been introduced to cloud data centres, offloading and accelerating specific networking applications and services [1][2][3].

As cloud data centre services are growing rapidly, the need and availability of "Hardware as a Service" (HaaS) [4], Software defined hardware programmability and Network Function Virtualization [5] [6] are introduced to provide infrastructure and service flexibility. The FPGA platforms promise flexibility in custom hardware design, parallelism and quick prototyping. FPGA based system advocates its need in data centre (DC) and software defined network applications due to a key set of well-known advantages. These include resource reuse for hardware configuration, acceleration, run-time (partial) reconfiguration. However, in these partial reconfigurable systems, FPGAs are usually under the control of a CPU that can either be on-chip or connected through a high-speed point-to-point interconnect such as PCIe or low speed interconnect such as JTAG. These create a critical dependency on deploying a CPU that is costly in terms of physical resources and power required, design complexity and limit on the configuration speed.

To provide FPGA as an independent resource that can be ubiquitously deployed and re-purposed on diverse environments from high performance Data Centres to low-cost low-power sensor networks, it must be installed as network-attached node and have the capability to be re-programmed over standard network protocols. As such, we propose a network based framework and reconfigurable model that sets the FPGA free from the attached CPU, Embedded CPU or point-to-point connectivity to a server. In our proposal, FPGAs can be re-configured as a standalone network-attached resource.

This work describes Reconfiguration over Network Protocol (REoN) framework to modify/update reconfigurable resources over the network by providing reliable transport of partial bit files. REoN includes a network stack to be supported by the network-attached remote FPGA (rFPGA) and the Software Defined Centralized Network Controller (SD-CNC). The protocol has the following attributes:

- Runs over a standard socket-based Internet protocol.
- Occupies a very small hardware footprint on the FPGA and it does not require an Embedded CPU or OS.
- Provides full access to partial reconfiguration, via Xilinx's Internal Configuration Access Port (ICAP).
- Supports reliable transport of partial bit-files with maximum throughput and minimum latency.
- It is independent of the other applications running on FPGA.
- It can be potentially applied to any FPGA type that supports Ethernet based I/Os.

The paper reports on the design and demonstration of REoN protocol. To the best of the authors knowledge REoN achieves the highest partial bit-file download throughput 505 Mbits/s and reconfiguration throughput of 696 Mbits/s while requires the least amount of resources.

The paper is organized as follows: section II reports on existing network based reconfiguration methods for FPGAs and provides the motivation to propose this new protocol. Section III explains the high level concept of the proposed work and its architecture. The design and implementation of REoN protocol and its functionality is presented and explained in section IV. Section V explains the test and measurement process and results followed by use of applications in section VI. Finally, section VII concludes the paper.

II. Background and Motivation

Few previous works have been conducted to introduce direct attached FPGA and partial reconfiguration in software defined hardware programmability in networks. Gordon et al. [5] introduce programmable hardware for implementing flexible SDN data planes and accelerating NFV functions. George et al. [6], Chen et al. [7] and Byma et al. [8], have used FPGA as virtual resource and partial reconfiguration via CPU connected over PCIe. The framework presented in [8] benefits users to deploy their own application in the FPGAs and access those applications over Ethernet. It has been shown in [8] that OpenStack service can be used to manage FPGA bit streams. Catapult [9] uses customized FPGA to accelerate page-ranking algorithms in Bing web search engine. It can achieve 90-95% improvement in ranking throughput for a fixed latency compared to the software approach. In [6] and [9], FPGAs are accessed and configured through the PCIe bus, whereas [7] uses a plain 1 GigE Ethernet connection and local embedded processor.

Network based communication frameworks have also been developed for partial reconfiguration [10] [11] [12] [13] [14]. An on-chip microprocessor has been used to manage reconfiguration of the device in [10], [12] and [13]. In case the systems are connected via a network, this wastes resources. Since the processor's main function is to implement a network stack, such resources can be used more effectively by other complex applications. The framework in [11] allows the transport of partial bit files from server over network with custom protocol headers into Ethernet payloads. This may reduce the latency due to the absence of higher layers however issues might arise when the FPGA belongs to IP subnet and coexists with other devices. Work presented in [14] suggests a network-attached framework. However, it does not provide a well-defined, complete, general and common framework for transporting partial bit files to an FPGA. Critically it doesn't support reliable transport of partial bit files. Furthermore, the writing configuration bit files process supports low speed interconnect (1 GigE) and offers one to one solution which is not the case with present network transport services and cloud data centres' requirements.

In contrast to those systems, we propose a framework which allows FPGA as a standalone resource for any Ethernet-based network environment. It is capable to deploy new physical or virtual network functions, data and signal accelerators, on demand over the network. Our approach also advocates custom hardware network stacks, which can be very fast and resource efficient. The centralized server is used to store the partial bit files as well as reconfiguration sequence by which partial bit files can be transferred directly to network enabled FPGAs.

III. High Level Architecture

A high level architecture for the proposed framework is depicted in Fig. 1(a). As mentioned in earlier sections, the implementation is divided in two aspects, the software part, running over the SD-CNC and hardware part, deployed on rFPGA platforms. The SD-CNC runs the Remote Reconfiguration and Standard Network Management (RR-SNM) System. The RR-SNM system is able to call an application specific function from a library through its software APIs and transfer its corresponding partial bit file to the required rFPGA. These APIs, as the part of the RR-SNM system have been provided to abstract low-level details of proposed protocol and provide user interaction to initiate configuration sequence. To do so, the RR-SNM APIs communicate with the Remote Reconfiguration Engine (RRE) on rFPGA via REoN protocol; a connection-oriented protocol over UDP which creates and reconfiguration session and guarantees order and delivery of partial bit file packets sent to the rFPGA from the SD-CNC.

RRE works as bridge between RR-SNM system and ICAP to manage the on-chip partial reconfiguration with local ICAP controller. UDP has been chosen as the base protocol because it is fast, simple, and its hardware implementation will occupy far fewer resources than more complex protocols such as TCP. The main targets of the proposed framework are:

- Provide direct interface to configuration port
- Full customized hardware network stack

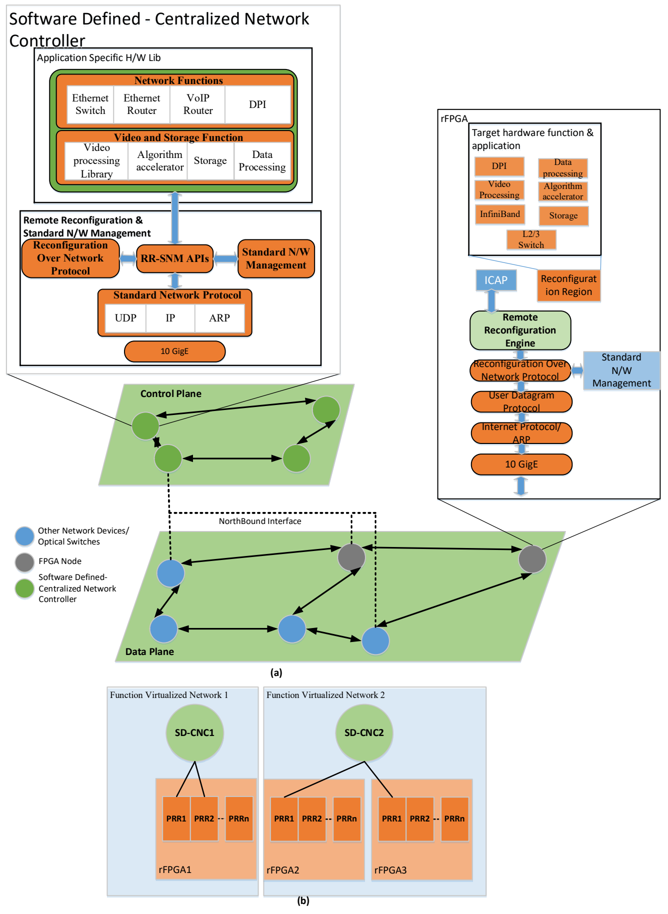


Fig. 1. (a) High Level Architecture for "REoN" Framework, (b) Logical Usecase for Proposed Framework

- Portable
- High throughput
- Low latency
- Optimized resource
- Simple server based resource and network management API

The logical use case for the proposed framework has been illustrated in Fig. 1(b). RR-SNM system running at one SD-CNC provides flexibility to re-program more than one partial reconfiguration region within same rFPGA or among different rFPGAs. This allows for FPGA-based virtual networks each controlled by separate SD-CNC and operating their own virtual network functions (VNFs) using a sub-set FPGA resources.

IV. Design and Implementaion

The targets for the proposed framework, mentioned in the previous section have a great dependency on each part of the proposed solutions. Aiming for low latency, high throughput, and minimal area requires the least-complex architecture, design and implementation. In this section, we will discuss how the REoN protocol has been implemented on both SD-CNC and rFPGA.

A. Protocol Specification

The most frequent and commonly used protocols in networks are UDP and TCP over IP. TCP is reliable but it consumes substantial resources. On the other hand, UDP needs less resources but it is connectionless and thus unreliable (or not guaranteed data transport). The network layer protocol (IP) is required in our solution to connect it easily in any type of network. For the transport layer, we required to find out a way in between TCP and UDP as we need both features to deliver partial bit files. One could propose a new transport layer protocol, which have simplicity like UDP and reliability like TCP. However, this would limit its widespread deployment since protocols and network drivers need to be re-written. The socket programming also would not be useful as it uses UDP/TCP/IP stacks. To address above difficulties, our solution introduces a new protocol REoN over UDP/IP, dedicated only to establish a reconfiguration session and transport partial bit files for rFPGA.

REoN protocol has been defined in such a way that rFPGA can share the same configuration channel among SD-CNCs within the control plane network. The REoN uses 49000/49001 as UDP port number. Any other packets from the same SD-CNC, using different UDP port numbers, would not be treated as REoN packets and forwarded to expected destination as in Fig. 1(a), running on the same rFPGA. To simplify the hardware implementation part of REoN, the protocol is designed for SD-CNC initiated communication only.

The REoN maximum packet size is 1472 (include 8 byte REoN header) following the Ethernet standard which is maximum of 1500 bytes (excluding Ethernet frame headers and FCS) including IP, UDP and REoN headers and payloads. There are two advantages to keep REoN packet in single Ethernet packet. One could be that the entire "REoN" packet can be encapsulated in a single Ethernet frame which limits data integrity issues. Second, there is no need to implement the packet fragmentation/defragmentation logic for IP protocol to reorder and reassemble IP packets that reduce IP protocol complexity which reduce the hardware utilizations. The IP protocol allows a packet to be marked "Don't Fragment."

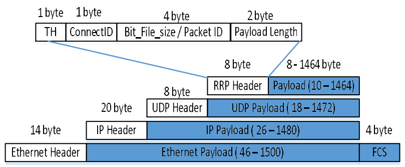
The UDP checksum has not been used and is replaced with zero to save rFPGA's resource utilization. The packet, sent and received over the network would be validated by CRC value calculated by Ethernet for the entire frame. Dependency over the Ethernet's CRC seems to be promising as each REoN packet will always be encapsulated on a single Ethernet frame.

DHCP protocol has not been used to get dynamic IP address to limit resource utilization. However, the static IP address for the rFPGA are fixed within the network. ARP has been adopted to make the proposed solution compatible with standard IP networks. This allows the SD-CNCs in the network to discover the rFPGA's MAC address. The rFPGA will reply on ARP request and store the connected SD-CNC's information like IP-MAC addresses pair. The rFPGAs does not need to send ARP request packet to the network as REoN is the host initiated protocol and the rFPGAs sends only acknowledgement (ACK) packets to the connected SD-CNC.

B. Packet Structure

ReON's header structure is shown in Fig. 2. The first byte of the REoN header provides the information about the type of handshake the packets belong to. REoN packets flowing from SD-CNC to rFPGA are always necessary to be request packet while in return rFPGA can only send ACK packet for each request to the SD-CNC. The connectID allows the configuration channel to be shared among various SD-CNC for the same rFPGA. By binding connectID and SD-CNC's IP address enables rFPGA to manage the partial bit files and keep separate from the partial bit files sent by other SD-CNC.

Four bytes followed by connectID have dual purposes and are used for packetID in request/ACK packet for bit file downloading and bit_file_size in request/ACK packet for partial bit file info handshake, successively. In other instances (different handshake purpose), these 4 bytes signify nothing and are set to zero. PacketID is used to place partial bit files in order for already acquired connectID. Failing that may raise very critical failure for rFPGA reconfiguration.



Type of Handshake	Function
00000000	Connection request
00000001	Connection ACK
00000010	BIT FILE INFO request
00000011	BIT FILE INFO ACK
00000100	Send BIT FILE
00000101	BIT FILE received ACK
00000110	Config Start request
00000111	Config Done ACK
00001000	Disconnect Request
00001001	Disconnect ACK

Fig. 2. REoN Header Structure

To send partial bit files, the RR-SNM API first packetize the partial bit files with maximum allowed REoN payloads which is 1464 bytes and then assigns packet ID to each packet. REoN payload for the last packet containing partial bit files may be equal or less than 1464 bytes. Payload length is always used to get the number of bytes in REoN payload.

C. Protocol Flow

The REoN flow is depicted in Fig. 3. The flow may be understood in phases, connection request (rqst) phase, partial bit files info rqst phase, partial bit file downloading phase, configuration (config) start phase and disconnect rqst phase. REoN has a well-defined message handshake to a) establish the connection (session) with rFPGA, b) send partial bit file's information c) send partial bit files, d) configuration start and e) tear-down the established connection (session). REoN will attempt five times to resend the continued handshake packets on failure to receive the ACK packet. After five attempted but failed handshake, RR-SNM API will clear associated information for connected rFPGA like connectID (if connection already established). Each request to reconfigure the rFPGA has certain flow and RR-SNM API will not send the next handshake packet until it receives the ACK packet for the previous handshake. However, this approach has not been used for partial bit files downloading packets. If the RR-SNM is waiting for an ACK packet after every bit file packet sent to rFPGA then overall throughput of the dedicated connection channel will be limited. A sliding window protocol is adopted to increase throughput while downloading partial bit files to the rFPGA. The window size of the protocol to each side is one, however a value greater than one at SD-CNC's side is possible.

Fig. 3(a) describes the ideal flow for the REoN, however, Fig. 3(b) illustrates the scenario where it might be possible that one of the packets carrying the partial bit file has been lost.

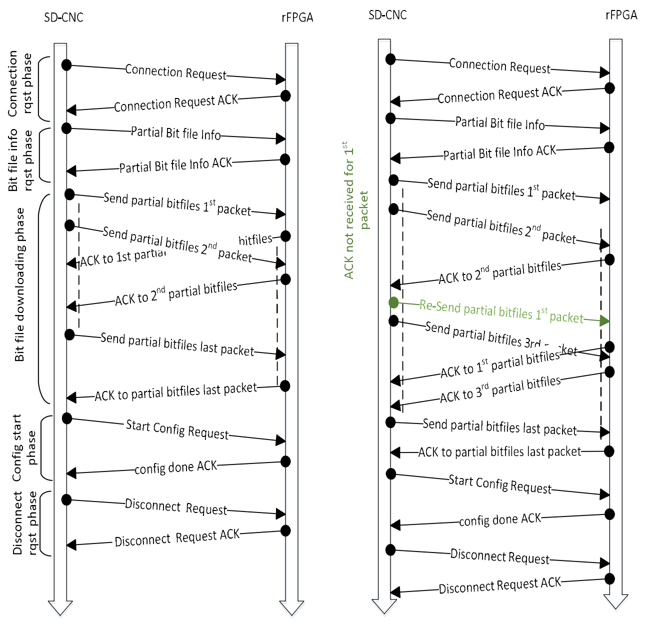

SD-CNC
rFPGA
Connection rqst phase
Bit file info rqst phase
Bit file downloading phase
Config start phase
Disconnect rqst phase
Connection Request
Connection Request ACK
Partial Bit file Info
Partial Bit file Info ACK
Send partial bitfiles 1st packet
Send partial bitfiles 2nd packet
ACK to 1st partial bitfiles
ACK to 2nd partial bitfiles
Send partial bitfiles last packet
ACK to partial bitfiles last packet
Start Config Request
config done ACK
Disconnect Request
Disconnect Request ACK
ACK not received for 1st packet
Re-Send partial bitfiles 1st packet
Send partial bitfiles 3rd packet
ACK to 1st partial bitfiles
ACK to 3rd partial bitfiles

Fig. 3. REoN flow. (a) Ideal Flow (b) ACK for bit files packet not received.

D. FPGA Design

Hardware design for the REoN has been depicted in Fig. 4. As discussed in previous sections that the main objective is to implement the required protocols and reconfiguration controller in hardware. In particular, the design must not include on-chip or off-chip CPU to achieve better performance with minimal resources.

Each network protocol like ETH, IP, ARP and UDP has been simplified by implementing separate TX and RX channels. ETH RX block is responsible to filter the incoming packets for own MAC address. If the incoming packets do not belong to MAC address, this block will not allow packet to be further processed. ETH arbiter block provides the usability of the REoN protocol through various Ethernet ports. However, this block could be optional if only one port is dedicated for the re-configuration. Standard Network Management block is used to store the network related information like IP address & UDP port address of each pair (SD-CNC and rFPGA). This block also filters the incoming packets by UDP port numbers. This identifies whether an incoming packet belongs to REoN or other user application such as standard network functions or data processing blocks implemented as partial reconfiguration model.

RRE appears as a core block for the REoN implementation. RRE receives the REoN header from the SD-CNC, decodes the handshake message and performs the following task. When RRE receives a handshake message to establish connection from RR-SNM API, it stores the requester SD-CNC's connectID. If connectID is not available, it assigns a new one and acknowledges the requester SD-CNC. RRE uses this information to filter incoming packets from all connected SD-CNCes and accepts only those packets from the connected SD-CNC to which the connectID has been assigned. RRE can support a maximum of 255 configurations connections; however current implementation allows only two SD-CNC to access ICAP at same time. RRE block also provides basic information of the partial bit files i.e. bit file size, to the ICAP controller that directly connects to ICAP port.

Partial bit files must be transferred to the ICAP port in order; failing this will trigger reconfiguration error. RRE manages the sequence of the partial bit files with each packetID number and place them in sequence to the local storage which is nothing but a dual port BRAM.

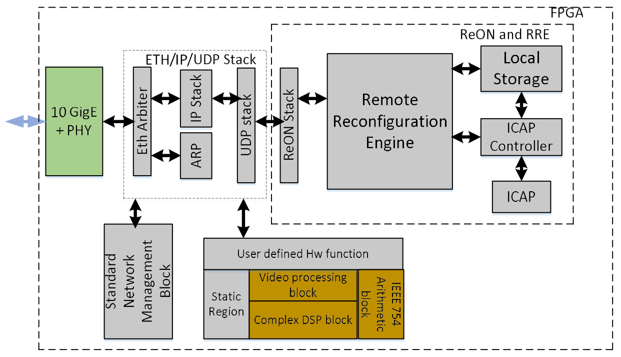


Fig. 4. Hardware footprint for REoN

In current experiment, 512 KB BRAM has been used which seems enough to store compressed partial bit files for complex design. The compression tool is an inbuilt feature of Xilinx Vivado tools, which reduces the size of the partial bit files for complex applications so that it can be stored in small BRAMs. BRAM size is not limiting the performance of the protocol and if the target application demands more storage, it can be increased up to required size. When RRE receives configuration start handshake message, it signals to ICAP controller to start reconfiguration. After reconfiguration is done by ICAP, it acknowledges the requester SD-CNC. RRE frees the occupied connectID once it receives connection (session) disconnect request from the connected SD-CNC.

V. Experiment, Performance and Analysis

To highlight the performance benefits of our approach over previous works, the key performance metrics have been measured. Table 1 gives some performance values for the existing work.

Myricom 10 GigE NIC card installed in standard Linux based system and NetFPGA SUME board have been used to test proposed framework. The current experimental set-up has been shown in Fig. 5.

Current experimental set-up combines both SD-CNC and data processing application in a single server. Network connection between server and rFPGA has been established over an Ethernet 10Gb/s link. However, the experiment can be further extended in data network. Our proposed implementation needs to achieve reasonably high throughput so that it may be useful across a wide range of applications

Most of the implemented logic operates at 156.25 MHz, which is the required frequency for 10GigE MAC. However, ICAP is running at 400 MHz, which provides 1.60 GBytes/s maximum local (ICAP to partial reconfiguration region) reconfiguration throughput. Incoming and outgoing packets have been traced and verified through Wireshark as shown in Fig. 6. Wireshark also illustrates the relative placement of the measured latencies and the protocol stack operations. We have characterized the throughput in two phases. First is the throughout between SD-CNC and rFPGA's RRE (download time) and the second between RRE and reconfigurable region through ICAP (reconfiguration time) as shown in Table I.

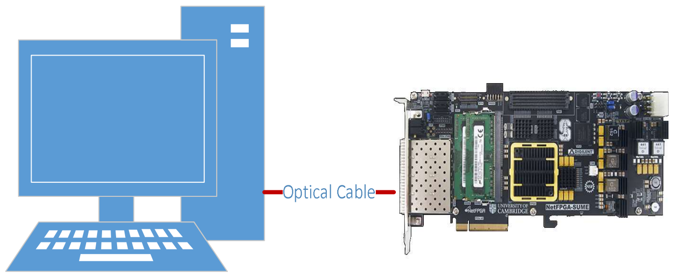
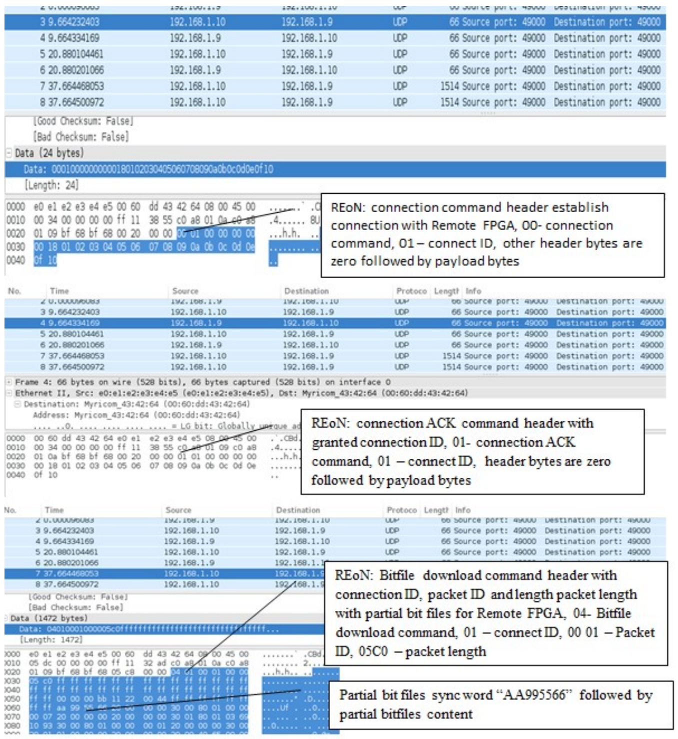


Fig. 5. Experimental set-up to test proposed framework



```
   2 0.000096083   192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   3 9.664232403   192.168.1.10   192.168.1.9    UDP    66 Source port: 49000  Destination port: 49000
   4 9.664334169   192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   5 20.880104461  192.168.1.10   192.168.1.9    UDP    66 Source port: 49000  Destination port: 49000
   6 20.880201066  192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   7 37.664468053  192.168.1.10   192.168.1.9    UDP  1514 Source port: 49000  Destination port: 49000
   8 37.664500972  192.168.1.10   192.168.1.9    UDP  1514 Source port: 49000  Destination port: 49000
      [Good Checksum: False]
      [Bad Checksum: False]
 Data (24 bytes)
    Data: 000100000000000180102030405060708090a0b0c0d0e0f10
    [Length: 24]

0000  e0 e1 e2 e3 e4 e5 00 60  dd 43 42 64 08 00 45 00   .......`.C
0010  00 34 00 00 00 00 ff 11  38 55 c0 a8 01 0a c0 a8   .4...... 8U
0020  01 09 bf 68 bf 68 00 20  00 00 00 01 00 00 00 00   ...h.h.  ..
0030  00 18 01 02 03 04 05 06  07 08 09 0a 0b 0c 0d 0e   ........ ..
0040  0f 10                                              ..
```

REoN: connection command header establish connection with Remote FPGA, 00- connection command, 01 – connect ID, other header bytes are zero followed by payload bytes

```
No.  Time            Source         Destination    Protoco Length Info
   2 0.000096083   192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   3 9.664232403   192.168.1.10   192.168.1.9    UDP    66 Source port: 49000  Destination port: 49000
   4 9.664334169   192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   5 20.880104461  192.168.1.10   192.168.1.9    UDP    66 Source port: 49000  Destination port: 49000
   6 20.880201066  192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   7 37.664468053  192.168.1.10   192.168.1.9    UDP  1514 Source port: 49000  Destination port: 49000
   8 37.664500972  192.168.1.10   192.168.1.9    UDP  1514 Source port: 49000  Destination port: 49000
 Frame 4: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
 Ethernet II, Src: e0:e1:e2:e3:e4:e5 (e0:e1:e2:e3:e4:e5), Dst: Myricom_43:42:64 (00:60:dd:43:42:64)
   Destination: Myricom_43:42:64 (00:60:dd:43:42:64)
     Address: Myricom_43:42:64 (00:60:dd:43:42:64)
     .... ..0. .... .... .... .... = LG bit: Globally unique ad

0000  00 60 dd 43 42 64 e0 e1  e2 e3 e4 e5 08 00 45 00   .`.CBd.
0010  00 34 00 00 00 00 ff 11  38 55 c0 a8 01 09 c0 a8   .4....
0020  01 0a bf 68 bf 68 00 20  00 00 01 01 00 00 00 00   ...h.h.
0030  00 18 01 02 03 04 05 06  07 08 09 0a 0b 0c 0d 0e   .......
0040  0f 10                                              ..
```

REoN: connection ACK command header with granted connection ID, 01- connection ACK command, 01 – connect ID, header bytes are zero followed by payload bytes

```
No.  Time            Source         Destination    Protoco Length Info
   2 0.000096083   192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   3 9.664232403   192.168.1.10   192.168.1.9    UDP    66 Source port: 49000  Destination port: 49000
   4 9.664334169   192.168.1.9    192.168.1.10   UDP    66 Source port: 49000  Destination port: 49000
   5 20.880104461  192.168.1.10   192.168.1.9    UDP    66 Source port: 49000  Destination port: 49000
   6 20.880201066  192.168.1.9    192.168.1.1
   7 37.664468053  192.168.1.10   192.168.1.9
   8 37.664500972  192.168.1.10   192.168.1.9
      [Good Checksum: False]
      [Bad Checksum: False]
 Data (1472 bytes)
    Data: 04010001000005c0ffffffffffffffffffffffffffffffff...
    [Length: 1472]

0000  e0 e1 e2 e3 e4 e5 00 60  dd 43 42 64 08 00 45 00   .......`.CBd
0010  05 dc 00 00 00 00 ff 11  32 ad c0 a8 01 0a c0 a8   ........ 2...
0020  01 09 bf 68 bf 68 05 c8  00 00 04 01 00 01 00 00   ...h.h.. ......
0030  05 c0 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff   ........ ......
0040  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff   ........ ......
0050  ff ff 00 00 00 bb 11 22  00 44 ff ff ff ff ff ff   ......." .D....
0060  ff ff aa 99 55 66 20 00  00 00 30 00 80 01 00 00   ....Uf . ..0...
0070  00 07 20 00 00 00 20 00  00 00 30 01 80 01 03 69   .. ... . ..0...
0080  10 93 30 00 80 01 00 00  00 01 20 00 00 00 30 00   ..0..... .. ...
```

REoN: Bitfile download command header with connection ID, packet ID and length packet length with partial bit files for Remote FPGA, 04- Bitfile download command, 01 – connect ID, 00 01 – Packet ID, 05C0 – packet length

Partial bit files sync word "AA995566" followed by partial bitfiles content

Fig. 6. Incoming and outgoing packets captured through Wirshark.

While sending partial bit files we have achieved throughput 505 Mbits/s, processing 8 bytes per clock, that even not utilizing full throughput offered. We may achieve more by increasing the window size for sliding window protocol running on SD-CNC. We have adopted two methods to measure latencies. The first method to measure latency is using Xilinx Vivado hardware debugger, running directly on hardware. The Ethernet subsystem, which includes ETH/IP/UDP/REoN as shown in Fig. 4, can receive maximum size packet 1500-bytes data packet, send an acknowledgement packet and its then ready to receive a new packet in 420 clock cycles or 2.6 µsec excluding MAC and PHY delays. It is important to mention here that while sending handshake packets other than downloading bit files, the Ethernet packet size is 66 bytes with all padding included.

The response time to the handshake packets has been calculated as 28 clock cycles, which is 128 nsec. The maximum partial bit files downloading throughput with 1500 data bytes per packet has been recorded as 505 Mbits/s for 10 GigE MAC operated at 156.25 MHz with 64-bit data width.

The latencies measured using the second method on the SD-CNC, includes APIs, operating system, and transport latencies. The average latencies measured with Wireshark have been found in between 96 µsec and 100 µsec in handshake phase (connection rqst phase, bit files info phase, config start phase and disconnect phase) from the rFPGA. For the bit file downloading phase, the two-way latency (up until SD-CNC gets acknowledgement that all bit files packets have been saved to local storage) dependents on the size of the bit file and has been given in Fig. 7.

TABLE I. COMPARISION OF PROPOSED WORK WITH EXISTING WORK

	FPGA Resources Utilization in %			Measurement		Interface	Embedded/ external CPU
				Downloading bit files	Reconfiguration		
	LUTs	FFs	BRAMs	Throughput* (Mbits/s)	Throughput* (Mbits/s)		
P. Bomel et al. [11]	NA	NA	NA	50	5	1GigE	PPC, uBlaze
O. Machidon et al. [10]	NA	NA	NA	NA	NA	1GigE	ARM
Byma et al. [8]	19	19	32	NA	NA	10GiE	External CPU
George et al. [6]	NA	NA	NA	25	1000	PCIe gen3	PCIE/CPU
Tze Hon Tan et al. [14]	11	10	23	NA	352	10 GigE	None
REoN (Our approach)	**2**	**1**	**8**	**505**	**696**	**10 GigE**	**None**

* Throughput measured here depends on maximum size of the partial bit files for each work presented.

The RR-SNM API's call to send the packet and initiate socket interface happens before the packet is seen by Wireshark. After the call to the socket interface, the OS's network stack prepares the packet for the network interface driver and has significant intervals. Wireshark captures the outgoing packet during this step. To measure the end to end latencies including OS and network driver delays, a software routine is written in each RR-SNM API. The end-to-end latencies recorded as average of 40 μsec for the handshake packets has been calculated by API provided functions which include delays for system call, OS and network drivers for both ongoing and incoming packets. Latencies measured at each level of framework to download partial bit files have been compared and depicted in Fig. 7.

Measurement of actual reconfiguration latency is done through Hardware Debugger only. It reflects the down time of the reconfiguration region. The latency has been measured in between the moment ICAP controller receives command to start reconfiguration process and the one that completes the reconfiguration process. The end-to-end latencies for RR-SNM APIs call for the reconfiguration phase have been also measured and compared with values captured by Wireshark and Hardware Debugger in Fig. 8. Reconfiguration latencies seen by RR-SNM API include OS overhead on the other side Wireshark includes MAC to MAC delay only. Separate REoN/UDP/IP/ETH hardware stack latencies for the configuration phase have been measured and shown in Fig. 8.

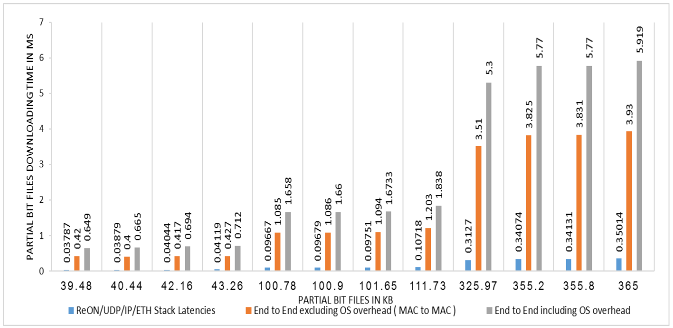


Fig. 7. Latencies measured at every stage while downloading bit file.

Reconfiguration latency highly depends on size of the partial bit files and how faster the ICAP is functional. We are able to clock ICAP at 400 MHz.

End to end latencies at each phase of proposed protocol have been depicted in Fig. 9. The latencies have been measured in μses and shown in logarithmic scale for larger partial bit files.

VI. APPLICATION USE CASES

Several applications in which proposed reconfiguration technique may be adopted, have been investigated with significantly large partial bit files and suitable for data-centre applications or any real time data transfer within other types of network. As a proof of concept, various processing algorithm has been chosen for real time raw video and data using the proposed framework model. Raw and uncompressed data transmission is used on post-processing applications and it needs high data volume transfer. An openCV platform has been used to send and receive raw RGB video over the network with UDP port number 49200 with rFPGA's IP and MAC address as destination. Video sent from host server has been captured by rFPGA and processed with reconfigurable video processing block and send back to server for display. UDP port number 49300 has been used to send raw data for data processing algorithm. REoN packets and other data packets have been differed by the UDP port numbers mentioned above. The video/data processing algorithms used in this experiment has been listed with their partial bit files size in Table II.

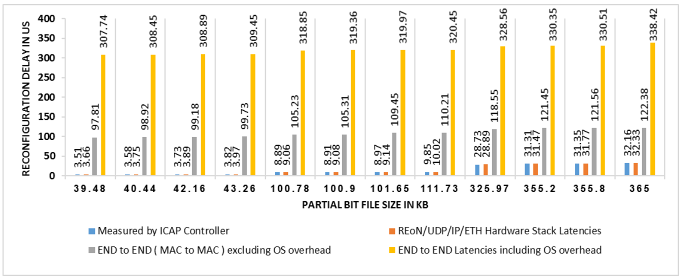


Fig. 8. Latencies measured at every stage while reconfiguration.

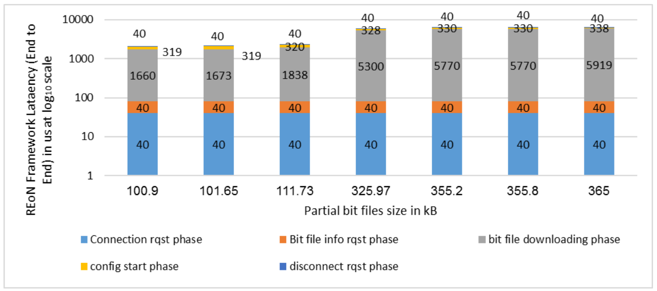


Fig. 9. End to end latencies at each phase of proposed protcol.

TABLE II. BITFILE SIZE FOR PARTIAL RECONFIGURATION MODEL

Partial Reconfigurable Modules (function)		Size (KB)
IEEE-754 Single Precision Arithmetic	Addition	100.78
	Subtraction	100.9
	Multiplication	101.65
	Division	111.73
Complex DSP Algorithms, 64 point Pipelined	FFT	325.97
	IFFT	355.2
	DCT	355.8
	IDCT	365
Video Processing Algorithms	Color Space converter	39.48
	Sobel operation	40.44
	Laplacian operation	42.16
	Gaussian operation	43.26

VII. CONCLUSION

A lightweight remote partial reconfiguration protocol, REoN is implemented over UDP/IP with minimal hardware resources. UDP/IP is used as the fundamental protocol, which not only allows FPGA to coexist in a standard network, but the RR-SNM APIs can use standard OS/Network libraries to send partial bit files. The proposed framework can be used to re-program high-speed network-attached FPGA-based sub-systems while eliminates the need of having an on-chip/on-board CPU. Our current version of framework achieves 505 Mbits/s partial bit files download throughput and reconfiguration throughput as 696 Mbits/s with almost negligible FPGA resources: 2% LUTs, 1% FFs and 8% BRAMs. Many applications targeting reconfigurable hardware, like NFV, reconfigurable L2/L3 switch, can benefit from this framework. It also validates the possibility that network algorithms or network sub-system mapped on FPGA can be swapped with sub-system for higher layer network protocol to upgrade the system on demand.

ACKNOWLEDGMENT

This work was supported by H2020 dRedBox project with grant agreement no. 687632 funded by European Commissions.

REFERENCES

[1] Putnam et al., “A reconfigurable fabric for accelerating large-scale datacentre services,” in Proceeding of the 41st Annual International Symposium on Computer Architecuture, ser. ISCA ’14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–24.

[2] J. W. Lockwood and M. Monga, "Implementing Ultra Low Latency Data Centre Services with Programmable Logic," 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, Santa Clara, CA, 2015, pp. 68-77.

[3] J. Weerasinghe, F. Abel, C. Hagleitner and A. Herkersdorf ,” Enabling FPGAs in Hyperscale Data Centres,” In proceedings of the IEEE International Conference on Cloud and Big Data Computing (CBDCom) Beijing, China, August 10-14, pp. 1078-1086, 2015.

[4] A. Stanik, M. Hovestadt, and O. Kao,“Hardware as a Service (HaaS): Physical and virtual hardware on demand,” In proceedings of the IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), Taipei, 3-6 Dec.2012, pp.830-836

[5] G. Brebner, "Programmable hardware in software defined networking," Optical Fiber Communications Conference and Exhibition (OFC), 2015, Los Angeles, CA, 2015, pp. 1-1

[6] G. S. Zervas, Q. Chen, and V. Mishra, "Network, Compute and Storage Function Programmability and Virtualization: An FPGA-based Disaggregated System," in Asia Communications and Photonics Conference 2015

[7] F. Chen et al., “Enabling FPGAs in the cloud,” in Proceedings of the 11th ACM Conference on Computing Frontiers, ser. CF ’14. New York, NY, USA: ACM, 2014, pp. 3:1–3:10

[8] S. Byma et al., “FPGAs in the cloud: Booting virtualized hardware accelerators with openstack,” in Proceedings of the 2014 IEEE 22Nd International Symposium on Field-Programmable Custom Computing Machines, ser. FCCM ’14, 2014, pp. 109–116

[9] A. Putnam et al., “A reconfigurable fabric for accelerating large-scale datacentre services,” in Proceeding of the 41st Annual International Symposium on Computer Architecuture, ser. ISCA ’14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–24.

[10] O. Machidon, F. Sandu, C. Zaharia, P. Cotfas and D. Cotfas, "Remote SoC/FPGA platform configuration for cloud applications," 2014 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM), Bran, 2014, pp. 827-832.

[11] P. Bomel, G. Gogniat, and J.-P. Diguet, “A networked, lightweight and partially reconfigurable platform,” in Reconfigurable Computing: Architectures, Tools and Applications, ser. Lecture Notes in Computer Science, R. Woods, K. Compton, C. Bouganis, and P. Diniz, Eds. Springer Berlin / Heidelberg, 2008, vol. 4943, pp. 318–323. 10, 13

[12] H. Tan and R. DeMara, “A Multilayer Framework Supporting Autonomous Run-Time Partial Reconfiguration,” Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 16, no. 5, pp. 504 –516, May 2008.

[13] B. Blodget, S. McMillan, and P. Lysaght, “A lightweight approach for embedded reconfiguration of FPGAs,” in Design, Automation and Test in Europe Conference and Exhibition, 2003, 2003, pp. 399 – 400

[14] Tze Hon Tan, Chia Yee Ooi and M. N. Marsono, "rrBox: A remote dynamically reconfigurable network processing middlebox," 2015 25th International Conference on Field Programmable Logic and Applications (FPL), London, 2015, pp. 1-4.