Reconfigurable hardware applications on NetFPGA for network monitoring in large area sensor networks

A. Belias[a,*], V. Koutsoumpos[a], K. Manolopoulos[a], C. Kachris[b]

[a] *NESTOR Institute for Astroparticle Physics, National Observatory of Athens, 24001 Pylos, Greece*
[b] *Networks and Optical Com. Lab, Athens Information Technology, Athens, Greece*

Abstract

A valuable functionality for sensor networks, distributed in large volumes is the capability to characterize and analyze the data traffic at wire speed and monitor the data prior to committing to permanent storage. As a demonstrator we use a reconfigurable hardware router for real-time monitoring of data before their transmission to further processing and storage. The reconfigurable hardware router is based on the NetFPGA platform. In this study we report on the hardware implementation to monitor web-based network applications and compare our results with a software based network analyzer.

Keywords: NetFPGA, Sensor Data Acquisition, FPGA system architecture

1. Introduction

In recent years, there was a rise in the interest to instrument large areas and volumes on land and in the sea with sensors from a variety of scientific communities. Notably, two large European projects, the European Multidisciplinary Seafloor Observatory (EMSO) [1] and the multi-km3 sized Neutrino Telescope (KM3NeT) [2], are in their preparation phase to deploy many distributed sensors covering very large observational volumes of order O(km^3), in several deep-sea locations around Europe and in the Mediterranean Sea respectively.

A common goal of these projects is to establish infrastructures to enable long-term (10+ years) monitoring of processes with interest to the bio- and hydrosphere in case of EMSO, and in the case of KM3NeT, the detection of Neutrino induced interactions in the Mediterranean deep-sea.

Regardless of optimisations of the platforms and networks for the specific operational needs of the deep sea environment, all sensor data will eventually have to be readout and transmitted to shore, be that through cabled or acoustics infrastructure.

We present a reconfigurable hardware electronics system capable to characterize and analyze the data traffic from sensors at wire speed prior to committing data to permanent storage. The number of data channels and their communications protocol can be reconfigured to adapt to a diversity of multiple sensors. Requirements to form spatial and temporal trigger patterns from a variety of sensors can be implemented as well as readout conditions of selective sensors.

The system hardware used is the NetFPGA 1G [3] which utilizes standard FPGA technology and common hardware components available off the shelf. Advantages of reconfigurable computing concepts have been outlined in [4], with an early implementation of a reconfigurable hardware system for deep-sea detectors in [5].

In the following we outline the hardware system used in this study, the architecture implemented for network traffic monitoring and report on preliminary results from tests made to compare with a data network analyzer realized in software.

2. Architecture

This study is based on the NetFPGA router architecture which has been expanded to monitor and measure traffic which is created by peer-to-peer file sharing torrent applications. To download a torrent file a client application downloads a file from a server application called tracker.

The tracker identifies the swarm (peers) which are computers on a network that either have the entire file or a part of it, or are in the process of sending or receiving it. The tracker helps the client software to trade parts of the file that the client has requested through other computers in the swarm. Hence, the client receives multiple parts of the file simultaneously.

The main problem in identifying the traffic that corresponds to torrents is that there is no specific TCP port for the bit-torrent applications. These applications are usually based on common ports such as the ports used by FTP

*Corresponding author. Tel.:+302723023300, Email address: anastasios.belias@cern.ch (A. Belias).

or HTTP applications. Therefore, it is necessary to perform a content-based inspection in order to identify that a packet belongs to a torrent flow.

We propose a design and its implementation on a reconfigurable hardware platform that performs at wire-speed deep-packet inspection in order to identify and characterize network traffic that belongs to bit-torrent applications. In our design we do not modify the packets (e.g. MAC address, IP address) we solely perform packet inspection and subsequently forward the packets on to their destination addresses on the LAN.

In the overall architecture our design is located transparently between the NetFPGA router and the clients that are located in a local area network (LAN).

In this design we can identify when a new torrent connection has been established. When this is the case a new entry is inserted into a table that records the torrent traffic based on the unique torrentŠs ID. On every packet that is received we perform a content-based inspection to identify packets belonging to a torrent connection. For each packet associated to a torrent connection we keep the unique torrentŠs ID and we add the size of the packet in a table.

Our design is implemented on the NetFPGA platform (Fig. 1). The NetFPGA card used in this study hosts a Xilinx Virtex II Pro FPGA, AND incorporates 4 Gigabit Ethernet ports, 4.5Mbytes SRAM and 64Mbytes DDR2 DRAM.

The NetFPGA card is supported through a GUI allowing access to the FPGA registers through the cardŠs PCI interface wit the host computer.

Using this NetFPGA card, we have implemented our design as a Network Analyzer system in hardware which filters packets containing an HTTP GET request or the string BitTorrent Protocol and sends the packetŠs information to the host computer through the PCI interface. On the host computer we make use of the GUI to search the hash table for the URLs and display them on screen. Figure 2 depicts the block diagram of the NetFPGA that has been implemented.

Figure 1: NetFPGA

The packets from a Gigabit Ethernet port are forwarded to the MAC FIFO. An input arbiter is used to extract the packets from the FIFO and forwards them to the Output Port Lookup. The Output Port Lookup is used to classify the packets and forward them to the correct output port. The Ethernet, IP LPM and IP Checksum are used in cases when the NetFPGA is used as a router. For our study, we have extended the reference design by adding three additional modules, the *http get filter* module, the *BitTorrent filter* module and the *Gnutella filter* module.

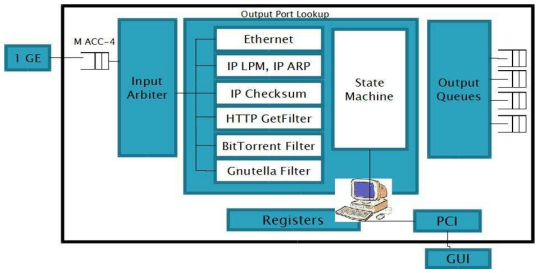

Figure 2: NetFPGA architecture

2.1. The http-get-filter module

The *http get filter* functions implemented in [6] are used for the URL extraction of the packets in order to identify that a packet belongs to HTTP traffic. According to [7], the HTTP protocol uses GET requests to send URL requests to a server. Packets containing a GET request are distinguished by containing the string "GET", at the beginning of the payload. In our module we first check the packet length to ensure that it is large enough to contain the string "GET". Next, the module checks the TCP protocol, which is used to transport HTTP and subsequently the destination port is inspected for HTTP port numbers (HTTP protocol, uses port 80). Finally, the TCP header length is checked since it varies in size for GET packets originating from different operating systems [7].

By checking the above mentioned protocol header fields and for the occurrence of the string "GET"at the beginning of the TCP payload, our module can identify the packets that belong to HTTP traffic. The module sends the packetŠs information to the host computer.

2.2. BitTorrent-filter module

The *BitTorrent filter* functions form a new pre-process block that is responsible for identifying packets that follow the BitTorrent protocol. The BitTorrent Protocol consists of a tracker and a peer. The tracker follows the HTTP protocol method in a similar way as with the *http get filter* module.

A tracker request contains the following parts: info hash, peer id, port and event. Info hash is a 20 byte SHA1 hash that contains the value of the info key in the Metainfo file [8]. Peer ID is a 20-byte string used as a unique ID for the client. Port is the port number that the client usually utilizes. Event, if specified, can be one of the following: started, completed, stopped or empty (which is the same as not being specified). If event is not specified, the request is repeated at regular intervals.

- Started: The first request to the tracker must include the event key with this value.

- Stopped: Must be sent to the tracker if the client is shutting down gracefully.

- Completed: Must be sent to the tracker when the download completes.

The handshake is a required message and must be the first message transmitted by the client.

The recipient may wait for the initiator's handshake; if it is capable of serving multiple torrents simultaneously (torrents are uniquely identified by their info hash). However, the recipient must respond as soon as it encounters the *info hash* part of the handshake (the peer id will presumably be sent after the recipient has sent its own handshake). The tracker's NAT-checking feature does not send the *peer id* field of the handshake.

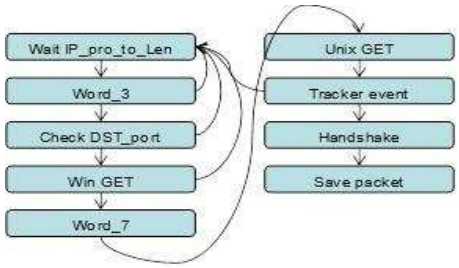

Figure 3: The BitTorrent Filter FSM

If a client receives a handshake with a *info hash* that it is not currently serving, then the client must drop the connection. If the initiator of the connection receives a handshake in which the *peer id* does not match the expected *peer id*, then the initiator is expected to drop the connection. Note, that the initiator presumably received the peer information from the tracker, which includes the *peer id* that was registered by the peer. The *peer id* from the tracker with the one in the handshake are expected to match.

In order to recognize a BitTorrent Protocol, first, we examine the HTTP GET request. Subsequently, we examine if the string "BitTorrent Protocol" is present in the packet. Further, we check if the length of this string is 19 bytes, and subsequently, we check if the tracker id equals the *peer id* and if the tracker *hash key* equals the *peer hash key*. After establishing that a packet is a BitTorrent packet, we save the packet in a hash table. In order to make the hash table, we use a hash function following the method cuckoo hashing [9]. The cuckoo hashing works in the following way. We use two different hash functions for the hash table. When we try to insert a new entry into the table and if the location is not empty, then we add the new entry to the specific location and then we move the old entry to the address created by the second hash function. This process is repeated until an empty location has been found.

The identification method for a BitTorrent packet is implemented by a state machine as shown in Figure 3. The state machine carries out a 10 stage elimination process for identifying a BitTorrent packet. The state machine initially idles in the *WAIT IP PROTO LEN* state, waiting for the IP packet length and protocol fields of the IP header to be present on the bus. The *preprocess control* signals the *BitTorrent filter* when this data is on the bus, and the elimination process is started.

The following states until the seventh state are the same as in *http get filter*, because the tracker uses the HTTP protocol. The eighth state is the *TRACKER EVENT*. In this state, we check if a torrent is started, stopped or completed. Then, in state HANDSHAKE, we check if the tracker id equals the peer id and if the tracker hash key equals the peer hash key. The process is concluded by saving the packetŠs information in the hash table.

2.3. Gnutella-filter module

The third application we have implemented in our design is the Gnutella module. Gnutella is a distributed file sharing application in which nodes, called *servents*, act both as client and server [10]. When a new servent connects to an existing Gnutella network, it has to establish a TCP connection with an already active servent. This servent broadcasts to its neighbors that a new servent is connecting.

For a servent to establishe a new connection with another servent, a packet is sent containing the "GNUTELLA CONNECT"string, Upon receipt, the second servent replys with the "GNUTELLA OK"string. This establishes a new connection between two servents. Once the connection is established, the nodes use the gnutella protocol in which the following commands are used:

- Ping: used to identify the active nodes

- Pong: the reply to the ping

- Query: The main mechanism for searching a file in the Gnutella network. If a servent receives a query and the query matches the files that are available, it replies with a QueryHit

- QueryHit: The reply to a Query. The packet includes important information about the reception of the packet.

The *gnutellafilter* has been implemented as an FSM that realizes the states as depicted in Figure 4. When the string "GNUTELLA CONNECT"is identified in a packet, then the state machine waits for the reply. If the reply packet contains the string "GNUTELLA OK"then a new gnutella connection has been established (CONNECT STATE) and the id is inserted into a hash table. The nest state waits until it receives the string "GET /get", in which a node is asking for a specific file. If a reply packet is received with an *HTTP 200 OK* field in the header, then the reception of a packet is initiated. The *gnutella filter* inspects the incoming packets and if they originate from an active gnutella connection, their size is added to the specific table and the packets are forwarded to the host computer.

3. Performance Evaluation

The design with the methods for the torrent filtering described in the previous sections has been implemented

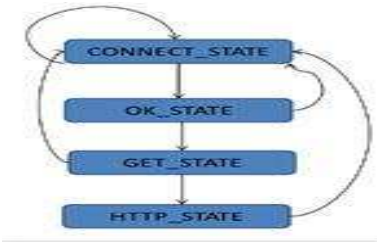

Figure 4: The Gnutella Filter FMS

Table 1: Logic resources of NetFPGA AppMon (VII Pro 50)

Type	Number	Percentage
Number of slices	11,131	47%
Number of Slice FF	9.378	19%
Number of 4-input LUT	16,547	35%
Number of BRAM	27	11%

on the Virtex II Pro FPGA of the NetFPGA card and consumes 209 slices and 8 BRAMs. Table 1 shows the resources of the implemented torrent filter. The minimum clock period of the design is 5.51ns (clocked at 117MHz). The databus of the proposed architecture is 64 bits and capable to process the network packets at wire speed. Therefore, it can achieve up to 7.5Gbps bandwidth (the maximum bandwidth supported by the NetFPGA is 4Gbps: 4x1Gbps).

To facilitate testing our designs we have developed the software interface to interact with the NetFPGA card. This interface is written in the ŞCŤ programming language and it reads packets from the PCI NetFPGA interface using raw sockets. Raw sockets allow packets to bypass the Linux TCP/IP stack and be handed directly to the application in the same form they were sent from the NetFPGA hardware. This software interface allows us to access the hardware registers of the NetFPGA design, which record the packet ids and sizes in of the aforementioned bittorrent modules.

To verify our design, first we verified our implementation in the simulation platform by using the Perl testing library. This library allowed us to create packets with specific TCP payloads, by first capturing GET packets using Wireshark [12] and then exporting the TCP header and payload using the *ExportasCArrays* feature and imported into our simulation scripts. Once verified in simulation, we created regression tests that mirrored our simulation tests. The regression tests were also created using the Perl testing library and allowed us to verify the operation of our design in real hardware.

Having verified the correctness of our implementation, we compared it with a software-based application monitoring framework called AppMon [11]. AppMon is an open-source, passive monitoring application for per application network traffic classification. It uses deep packet inspection to accurately attribute traffic flows to the applications that generate them, and reports in real time the network traffic breakdown through a GUI.

In our setup we use a host computer with the Fedora 12 operating system on an AMD dual core processor running at 1.0 GHz, with 4GB of RAM, and an Intel Pro/1000 Dual-port NIC. In these tests we are generating real-time traffic, by running a bit-torrent application in order to capture HTTP and BitTorrent packets. We recorded the CPU utilization of the host computer when the packets were analyzed by the AppMon and when they were analyzed by the NetFPGA analyzer using one 1Gbps Ethernet port. The results showed that in the case of the software-based application monitor, the CPU utilization lies between 80%-100%, while when using the NetFPGA analyzer the CPU utilization is typically about 20%. In the tests with our NetFPGA analyzer most of the CPU utilization is due to the communication through the PCI interface and the processing for the GUI.

4. Conclusions

In this study we presented a hardware accelerated Network Analyzer system that performs traffic filtering and packet identification based on the readily available NetFPGA platform. As proof of concept, the proposed scheme has been configured to analyze HTTP, BitTorrent and Gnutella Protocols; however it can be easily expanded to support any protocol.

In large scale distributed sensor networks one is faced with the challenges of monitoring and classifying the network traffic from a variety of different sensors. As shown in this study, reconfigurable hardware systems, such as the NetFPGA, facilitate greatly to classifying and monitoring of network traffic with a variety of protocols. Further studies are planned to demonstrate the use of the NetFPGA platform in actual sensor network applications.

References

[1] EMSO, http://www.emso-eu.org
[2] KM3NeT Consortium, http://www.km3net.org
[3] www.netfpga.org
[4] A. Belias, "Reconfigurable computing concept for the on-shore data acquisition system of a km3-scale underwater neutrino telescope", Nucl. Instr. and Meth A 602 (2009) S143-S145.
[5] E.G. Anassontzis et al. "Commodity readout electronics for an underwater neutrino telescope", Nucl. Instr. and Meth A 602 (2009) S140-S142.
[6] M. Ciesla, et al. "URL Extraction on the NetFPGA Reference Router", Technical Report, www.ee.unsw.edu.au.
[7] A. Callado, et al. "A Survey on Internet Traffic Identification", IEEE Communications Surveys and Tutorials, vol. 11, no.3, pp.37-52, 2009.
[8] The BitTorrent protocol, http://www.bittorent.org
[9] R. Pagh, F. F. Rodler, F. Friche, "Cuckoo Hashing", doi:10.1.1.25.4189.
[10] I. Ivkovic, "Improving Gnutella Protocol: Protocol Analysis And Research Proposals", Technical Report, 2001, www.cs.cornell.edu
[11] Appmon, www.ipworx.com/product/appmon/manual.pdf
[12] Wireshark, http://www.wireshark.org/