A 10GE Monitoring System

Ariën Vijn
arien.vijn@ams-ix.net

November 2006

Abstract

Capturing network packets is a valuable technique for troubleshooting network problems. Especially when the troubleshooter has to deal with network elements that are not under his or her control.

Capturing at wire speeds up to one gigabit per second is feasible with fast general-purpose computer hardware. But that hardware is too slow for ten gigabit per second ethernet (10GE). Specialised hardware is required for that.

This paper describes the modification of a commercially available 10GE networks security system, into a network analyser.

1 Introduction

Internet is a network of networks. This cliche implies that these different networks need to be interconnected, somehow, somewhere. For cost-saving reasons, interconnections are often realised at central points known as internet exchanges. Technically an internet exchange (IX) is often a set of switches (switch park) as schematically pictured in Figure 1.

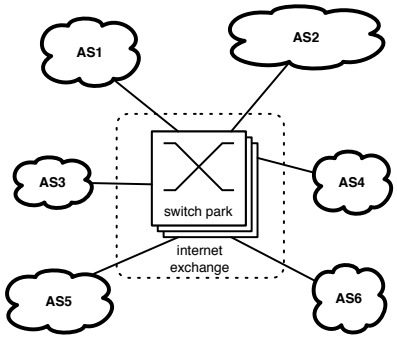


Figure 1: Interconnecting networks

This switch park only facilitates the exchange of traffic, each network has to make arrangements with the others for doing this.

Troubleshooting network problems at an internet exchange involves at least three parties. Namely, the internet exchange operator and two or more participants, each with their own systems, know-how, procedures and culture. One can imagine the extra challenges that may arise if the interconnection is not working properly.

Passive monitoring - watching the traffic as it passes by - has proven to be an essential technique to determine and isolate network faults in the path between different networks.

Passive monitoring at speeds less than 1 Gbps can easily be accomplished using fast general purpose computer hardware. Numerous open source applications, mostly based on libpcap [1] have been created to do this. But, ten times faster, at 10 gigabit per second it is another game. The minimum time in-between frame is just 9.6 ns, Figure 2 illustrates this. Special hardware is required to do that at line rate.

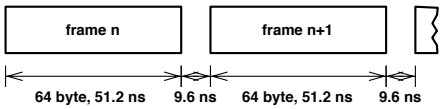


Figure 2: Minimum timings 10GE data stream

The P10 IDS[1]/IPS[2] appliance from Force10 networks [2] contains such hardware in the form of a special purpose 10 Gigabit Ethernet (10GE) Network Interface Card (NIC). This paper describes this NIC and the modifications required to turn it into a network analyser system to determine and isolate network problems.

[1] Intrusion Detection System
[2] Intrusion Protection System

2 System Overview

The P10 is an appliance built around a generic PC server equipped with special purpose NIC. The appliance is typically placed in-line, as is shown in Figure 3. The box acts as repeater that is inspecting the traffic passing through it.

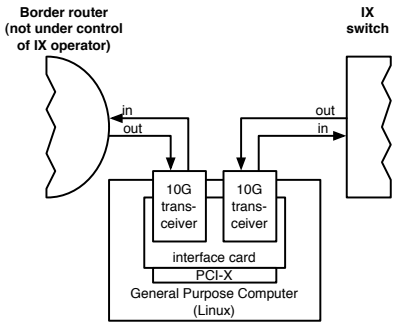


Figure 3: Appliance in-line

The purpose of the special NIC is to reduce the data rate to levels that can be handled by the host. It does that by inspecting all data at line-rate and copying only the 'interesting' frames to the host.

Figure 4 shows an overview of the maximum data rates that can be processed by the various components. Forwarding between the transceivers can be performed at full duplex 10GE line rates. That's a maximum frame rate of more than 14 million frames per second in each direction.

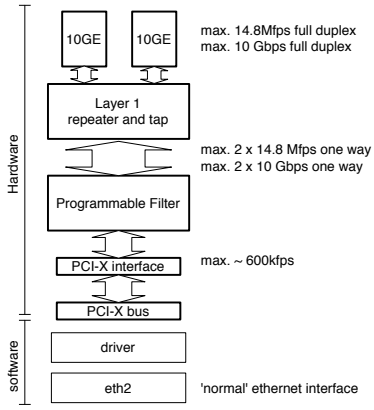


Figure 4: Performance

From that stream, a maximum of about 600,000 frames per second can be transferred to the host via the PCI-X bus. The filtering of the two data streams (one in each direction) is performed at full line rates.

From a host perpective the card acts just like a normal ethernet interface. This is realised via kernel modules or driver software. So all open source applications based on libpcap can be utilised to process that filtered traffic stream. However, it is a receive-only interface, it cannot actually transmit anything.

2.1 Block Diagram

This Section describes the NIC, which is almost entirely built around programmable electronic components. Namely: two so called Field Programmable Gate Arrays (FPGA) and a Complex Programmable Logic Device (CPLD). Especially the FPGAs make it an attractive object to modify for different purposes other than the security application it was originally designed for.

Figure 5 shows the functional blocks of the NIC.

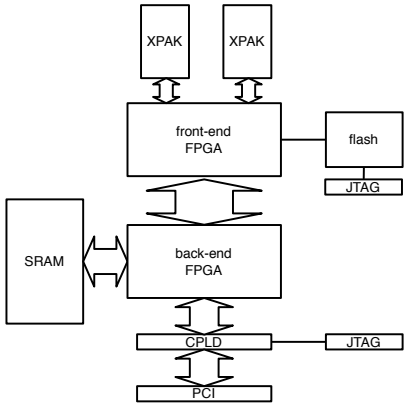


Figure 5: Block diagram of P10 NIC

XPAK trancievers: These are the interfaces for the data stream to be monitored. The electrical interface of XPAK transceivers is the standardised XAUI[3] interface. This standard defines four data lanes per direction. Each lane can transfer up to 3.125 Gbit/s of 8B/10B encoded data. These interface directly with the front-end FPGA, using the readily available interface solution from the FPGA manufacturer. [3]

[3] Pronounced as "Zowie".

Frontend FPGA: This chip forms the layer one forwarding engine, or repeater, between the two XPAK transceivers. The forwarded data is also copied (tapped) and prepared for further processing by the back-end FPGA.

Backend FPGA: This FPGA is the heart of the system. It filters the data it gets from the front-end FPGA to reduce the data rate. Frames that match the filter expression get passed through to the CPLD (see next item) so they can get processed by the host. This FPGA is reprogrammable from the host to apply new filter expressions.

CPLD: The PCI-X host interface is realised in this chip. It can act as PCI target or PCI bus master and it is also capable of transferring packets directly into the host memory via direct memory access (DMA).

SRAM: Fast memory to keep state[4] and fragments[5]. This functionality is not really needed in a network analyser.

JTAG: CPLD and the front-end FPGA can only be programmed via an external JTAG programmer. These connectors might also be used for debugging purposes.

3 FPGA Firmware

This Section describes the two FPGAs in more detail and emphasis on changes and additions made to convert the P10 from a network security appliance into a network analysis tool.

3.1 Front-end FPGA

This FPGA loads its code from flash at the the time the card is powered up. So this FPGA depends only on the power it gets from the host computer. This means that a reset or a system crash of the host does not affect the forwarding of data between the two transceivers.

[4]For stateful inspection.

[5]Required to hold fragmented packets until they can be assembled for inspection.

The front-end FPGA contains two instances of the XAUI code to convert the 10Gbit/s full duplex signal to two 64 bits wide buses, one for each direction. The bus interfaces are connected back-to-back via AND-gates. These gates may be used to interrupt the flow of data when the card is used as IPS. For network monitoring purposes this blocking function is not required.

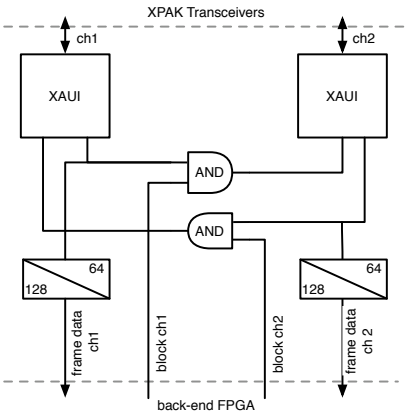


Figure 6: Front-end FPGA

The connection between the two XUAI blocks is tapped. The tapped data is then fed into a preprocessor[6] when its used as IDS/IPS. The purpose of this pre-processor is to normalise the data stream, so it can be processed by the back-end FPGA in a uniform manner. For example it removes 802.1Q (VLAN) headers if present, so the back-end FPGA does not have to take care of variable header positions.

This preprocessor function had to be removed or bypassed for monitoring purposes, since it alters the content of the frame which is obviously undesirable for that application.

At egress, the data is converted from 64 to 128 bits to reduce the clock frequency on which the data is processed by the back-end FPGA.

3.2 Back-end FPGA

As mentioned before, this is the heart of the system. This FPGA is programmed as a so called Multiple Instruction Single Data (MISD) system, which is one of the four architectures described in Flynn's Taxonomy [4]. It is by far

[6]not shown in Figure 6.

the least commonly used form of parallel computing but it is very suitable for this purpose, as there is always only one ethernet frame at one time to inspect against many rules. Figure 7 shows how this is realised.

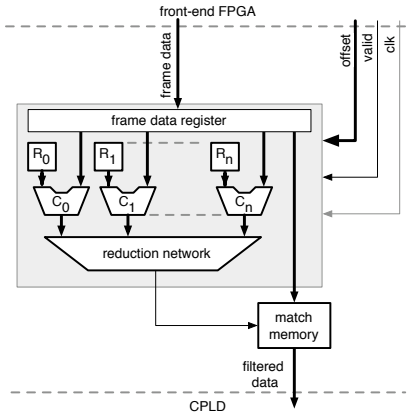


Figure 7: Back-end FPGA

Each portion of the ethernet frame received from the front-end FPGA is fed to many blocks of combinational logic (C). Each block compares the data against a pre-defined byte pattern (R). One or more blocks form one instruction. In other words, all instructions are executed in parallel over one single piece of data. Whether a particular instruction is applicable depends on the offset counter that comes from the front-end FPGA.

Once the data is flagged as valid by the front-end FPGA, the results of all instructions are taken from the reduction network. That result determines whether the frame should be copied to the CPLD so it can be transmitted to the host over the PCI-X bus. Besides that, the results can be utilised to interrupt the forwarding of data in the front-end FPGA if the system is operating as an IPS[7].

This architecture can be used for network monitoring purposes as well. But there are some differences between a network analyser and a ISD. These differences are described in the next Section.

[7]This is not shown in Figure 7.

4 IDS vs Network Analyser

IDSes are fairly static monitoring systems, checking the data stream against a lot of known patterns. New attacks mean new patterns, while the old onces remain valid. The P10 benefits from this fact by compiling known patterns in the FPGA bit code. At compile time, the synthesizer will optimise to the least amount of FPGA resources, within predefined timing restraints. Rules entered at compile time are called static rules.

Adding patterns at run time (dynamic rules) requires registers to hold any possible pattern for any possible offset. Hence dynamic rules take considerable more FPGA resources than static rules. Network analyses requires almost only dynamic rules because that work is typically conducted by adjusting filter expressions to narrow down the problem.

Another difference lies in the nature of filter expressions. An IDS typically matches many different rules, while ignoring the rest. That results in simple boolean function:

$$f(IDS) : R_0 + R_1 + \ldots + R_n$$

While rules (R) consist of one or multiple patterns (P):

$$R = P_0 \cdot P_1 \cdot \ldots \cdot P_n$$

Please note that there is no need for a NOT-operator (corner cases aside).

In network analysis the filter expressions are not so uniform. Unlike those of IDSes, they very often contain negations. Because uninteresting patterns can often be matched and the interesting ones often not. Therefore a NOT-operator was added to the instruction set.

Another problem are the frequent use of parentheses in filter expressions done for network troubleshooting. The current reduction network does not support any parentheses, so these have to be removed from the filter expression, before entered into the hardware, using boolean math. This means that same patterns have to be entered in multiple registers.

Currently, work is in progress to accomodate parentheses in hardware. But it will only allow a limited level of nested parentheses. Each level adds to the time the reduction network needs. This time is limited to the clock cycle on which the data is flagged valid. This part of the work is all about finding a balance between expanding expressions using software and the available resources in hardware.

The P10 does pattern matching per octet. But network analysis is also about matching one bit while the remaining 7 bits can be any combination. For example filtering on the multicast bit in a MAC address. In the original IDS code this would mean 128 matching patterns. That is not so much of a problem for static rules that are given at compile time, because the synthesizer will optimise the number of gates needed.

It would be very inefficient, if not impossible, to do bit-wise matching at run time. Simply because the amount of resources required for this one operation. This has changed by adding a bitmask mechanism to cover the remaining bits.

Last but not least, as layer-2 internet exchange operators are mostly interested in issues at the datalink layer. The original versions of the back-end FPGA firmware were not able to process the first 128 bits of the ethernet frame[8]. That's because an IDS operates from layer-3 to 7, perhaps even layer-9 [5]. Hence it was not needed to process the first part of the frame, although it gets presented to the back-end FPGA. This was first changed in our own firmware and is now an option in the stock firmware as well.

4.1 New Functionalities

The functionality described so far is in essence not different from the original IDS/IPS system. For the purpose of network analyses a number of new functions have been added.

Programmable counters Counting events is often just as useful as getting all details of each frame. So instead of triggering the match memory (see Figure 7) a counter value is raised. The counter values can be zeroed from the host, which is particularly handy during troubleshooting. Currently these counters can only count frames, not frame sizes or frame rates.

Sampling A function is added to get a random sample of frames at a continuously adjustable frame rate to the host. This is useful when the exact nature of the issue is unknown. Empirical analysis showed that the behaviour of the card is very predictable. To avoid systematic errors a pseudo random generator was added.

Checksums For analysing packet corruption, it is essential to get the frames containing corrupted packets. Please note that the frame check sum (FCS) is not helpful here since this is calculated on egresses by its source.

At this point we can only calculate IPv4 checksums of IPv4 packets without options. TCP, UDP and ICMP checksums can not easily be calculated in the back-end FPGA because the end of the frame is unknown there. That information got lost in the front-end FPGA. A possible workaround would be the usage of packet and header length information out of the packets it is inspecting. But that seems a bit dodgy solution. Better and cleaner is to move the checksumming functionality in the front-end FPGA. In there, it will also be possible to detect frame and packet size mismatches.

Ring buffer Sometimes it is useful to capture frames transmitted before and/or after a certain condition was met. This is a good use of the SRAM present. However no work has so far been conducted to realise this functionality. This feature would require a considerable rewrite of the a large parts of the readily available building blocks, which are briefly described in the next Section.

5 API

All modification were possible due to the fact that the firmware of the back-end FPGA is ei-

[8] starting from the preamble

ther available as source code or in binary libraries. These 'building blocks' , commonly used functions (like packet processor, host interface and memory management) are needed to support the MISD-machine.

The signals of these blocks are documented to allow end users to program their own functionality. Figure 8 is visualising the position of that new function and the predefined building blocks with common functions.

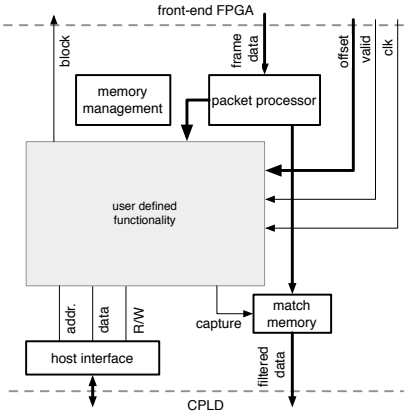


Figure 8: User defined functionality

With the description of signals skilled users, can program their own modules in Verilog [6]. That module connects to all the available signals. From there on it is up to the user how to use these signals. This user defined module gets compiled in the firmware instead of the MISD-machine. But in practice, the MISD model is followed in our own modules too.

6 Software

The software part can be divided in two pieces, the kernel modules that form the driver for the NIC and user software to manage rules. We use unmodified kernel modules.

The user application has a curses interface to show counters and to manage dynamic and static Snort rules [9]. For static Snort rules, the Xilinx ISE suite [8] is called to compile the bit code for the back-end FPGA.

[9]Snort [7] is a software based IDS system. Its rule-syntax is used to define rules for this hardware based system.

We aim for a CLI application that parses libpcap filter expressions. That has not been realised yet as it is still a matter of research on what can be done in hardware and what needs to be done in software as mentioned in 3.2.

Currently we operate by creating a hex dump to be loaded in the registers of the back-end FPGA. Which is of course not a real user interface.

7 Monitoring via a PXC

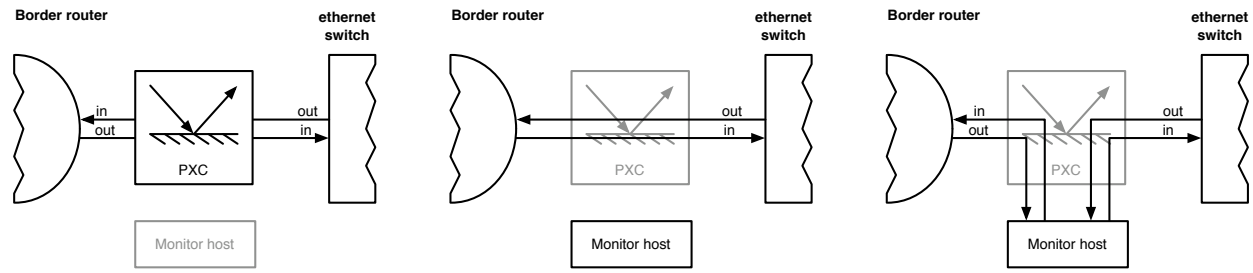
This Section is not really about the 10GE capture card itself, but rather about the way it is being utilised.

Typically monitoring is done via the mirror port feature of ethernet switches. It copies all data from and/or to the to-be-monitored interface to a designated port. This designated port, the mirror port, connects to analyser equipment. In fact data duplication takes place 'somewhere' in the ethernet switch. Any problem that arises after the duplication is not copied to the mirror port. Duplication also means that the bandwidth usage doubles, or in case of full duplex monitoring triples in one direction. Internal data paths might get saturated because of that fact.

The P10 is designed for permanent in-line use to protect networks. It does not have to depend on a mirror port and that has its benefits. But putting the card in-between a to-be-monitered link requires an interruption of that link, which is a real drawback.

This work is conducted for the Amsterdam Internet Exchange (AMS-IX) [9] were all 10GE member connections are made via so-called photonic cross connects (PXCs) as shown in Figure 9a. These devices, made by Glimmerglass [10] connect the light out of one fiber into any other fiber via moveable mirrors. The data does not undergo any electrical conversion.

PXC devices are sort of remotely controllable patch panels. Changes take typically less than 20 ms plus the time connected devices require to recover from the short loss of light, which varies but is typically less than 1 second. Such



(a) Place of PXC (b) Normal operation (c) Monitoring in-line

Figure 9: Use of photonic cross connect

short interruptions are considered acceptable. If they do not take place frequently.

The PXCs can therefore easily be used to put the monitor in and out as shown in Figures 9c and 9b. It also provides enough flexibility for other setups, like the use of mirror ports.

8 Conclusion

The P10 contains all components to become a powerful network analysis tool. The hardware is reprogrammable, the source code of most components in the firmware is available. As is the source code of the accompanying software.

An IDS and a network analyser are not the same. They have a lot in common but an IDS is a very large but uniform filter expressions. Filter expressions used for network analysis are dynamic and far from uniform. Network analysers require new special functions like checksumming. Some extra functionality is already realised, some of the existing properties required changes.

Not all functions have to be implemented in the firmware. The special hardware only has to reduce the data rates to levels that can be handled by existing software.

9 Acknowledgments

I would like to express my gratitude: To Elisa Jasinska, Henk Steenman, Niels Bakker and Robin Bakker for their time and efforts to review this paper; To Livio Ricciulli for his support and contributions to this project; To the entire team of AMS-IX for their support, trust and patience.

References

[1] Libpcap
`http://www.tcpdump.org`, 2006.

[2] Force10 Networks P-Series
`http://www.force10networks.com/products/p-series_overview.asp`, 2006.

[3] XAUI - 10GE Attachment Unit Interface
`http://www.xilinx.com/esp/wired/optical/xlnx_net/xaui.htm`, 2006.

[4] Multiple Instruction Single Data
`http://en.wikipedia.org/wiki/MISD` 2006. 1972.

[5] 9-Layer OSI Model
`https://secure.isc.org/index.pl?/store/t-shirt/`, 2006.

[6] Verilog hardware description language
`http://en.wikipedia.org/wiki/Verilog`, 2006.

[7] Snort
`http://www.snort.org`, 2006.

[8] Xilinx ISE
`http://www.xilinx.com/products/design_resources/design_tool/index.htm`, 2006.

[9] Amsterdam Internet Exchange
`http://www.ams-ix.net/`, 2006.

[10] Glimmerglass
`http://www.glimmerglass.com`, 2006.