Time Synchronization Solution for FPGA-based Distributed Network Monitoring

Ferenc Nandor Janky and Pal Varga

***Abstract*—Distributed network monitoring solutions face various challenges with the increase of line speed, the extending variety of protocols, and new services with complex KPIs. This paper addresses one part of the first challenge: faster line speed necessitates time-stamping with higher granularity and higher precision than ever. Proper, system-wide time-stamping is inevitable for network monitoring and traffic analysis point of view. It is hard to find feasible time synchronization solutions for those systems that have nation-wide, physically distributed probes.**

Current networking equipment reside in server rooms, and have many legacy nodes. Access to GPS signal is complicated in these places, and Precision Time Protocol (PTP) does not seem to be supported by all network nodes in the near future – so high precision time-stamping is indeed a current problem. This paper suggests a novel, practical solution to overcome the obstacles.

The core idea is that in real-life, distributed network monitoring systems operate with a few, finite number of probe-clusters, and their site should have a precise clock provided by PTP or GPS somewhere in the building. The distribution of time information within a site is still troublesome, even within a server rack. This paper presents a closed control loop solution implemented in an FPGA-based device in order to minimize the jitter, and compensate the calculated delay.

***Keywords*—network monitoring, time synchronization, hardware acceleration, closed control loop**

I. INTRODUCTION

Network monitoring has a well-established practice at telecommunication operators. There are fundamentally different solutions available – depending on what kind of data are initially available and how they are gathered. The least flexible solutions are based on the functional networking elements: they can provide pre-digested reports, statistical counters, and occasionally (when not under heavy load), even detailed information on the actual messages. Some operators use standalone protocol analyzers, which do not suffer from the temporal, load-related bottlenecks – rather, they have spatial data capture issues: only a segment of the network is visible at any given time. On the other hand, complete traffic information can be gathered by network-wide traffic monitoring. These latter solutions are based on passive, distributed probes; central processing entities; and client software – also distributed – at the operating personnel. This paper discusses a peculiar problem of such systems: effective time synchronization among the entities.

The authors are with the Department of Telecommunications and MediaInformatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Magyar tudósok körútja 2., 1117 Budapest, Hungary (phone: +36704213213; e-mail: fecjanky@gmail.com and pvarga@tmit.bme.hu)

DOI: 10.36244/ICJ.2018.1.1

Network traffic analysis requires the understanding of the order of the messages appearing in the network, even if they appear at different interfaces. This makes high resolution and high precision time-stamping the basic requirement, beside lossless message capture. While there are standardized network protocols available for tackling this issue, there are practical obstacles in their network-wide usage. Although the Network Time Protocol (NTP) is widely available [1], it cannot be used as a general purpose synchronization protocol. In fact, the message transfer delay between NTP clients and servers is not compensated, hence the different nodes end up setting their local time to a clock value with a random delay. The typical order of the forwarding delay in current core routers is in the 0.5-5 microseconds range, depending on the traffic volume – among other factors. Since the minimum packet interarrival-time is 0.672 microsecond even at a 1 Gbps link (and 67.2 nanoseconds for a 10 Gbps link), such delays cannot be left without compensation for time synchronization.

Precision Time Protocol (PTP), on the other hand covers the delay-compensation issue well [2]. Unfortunately, PTP is not at all wide-spread, even after 10 years of commercialization for PTPv2. The concept, however, necessitates that all network nodes in the path have PTPv2 capability. Otherwise – even if one node cannot compute and share its delay data –, compensation of time information is not possible.

Another solution could be to introduce time information of GPS (Global Positioning System) satellites into the nodes – this is not feasible, since rack cabinets in server rooms lack the line of sight.

We can suppose that at least one machine at each monitoring site has the possibility to get synchronized to the master clock of the network (e.g. through PTP or GPS). Nevertheless, synchronizing all clocks within the site with nanosecond-range precision, is still a challenge.

This paper presents a solution for the time synchronization issues of systems with FPGA-based monitoring probes. What makes FPGA a key player here is that hardware-acceleration removes the jitter of operating system and protocol-stack delay from the equation. The delay of handling time information within an FPGA is constant, we can calculate with it precisely – and compensate this delay for the time-stamp.

In this paper we focus on the time synchronization challenges of a monitoring site. The implemented solution is based on the practical pre-requisite that each site has a reference clock available for the monitoring system. This paper suggests an FPGA-based clock synchronization method for the distributed monitoring equipment, more precisely, its interface cards.

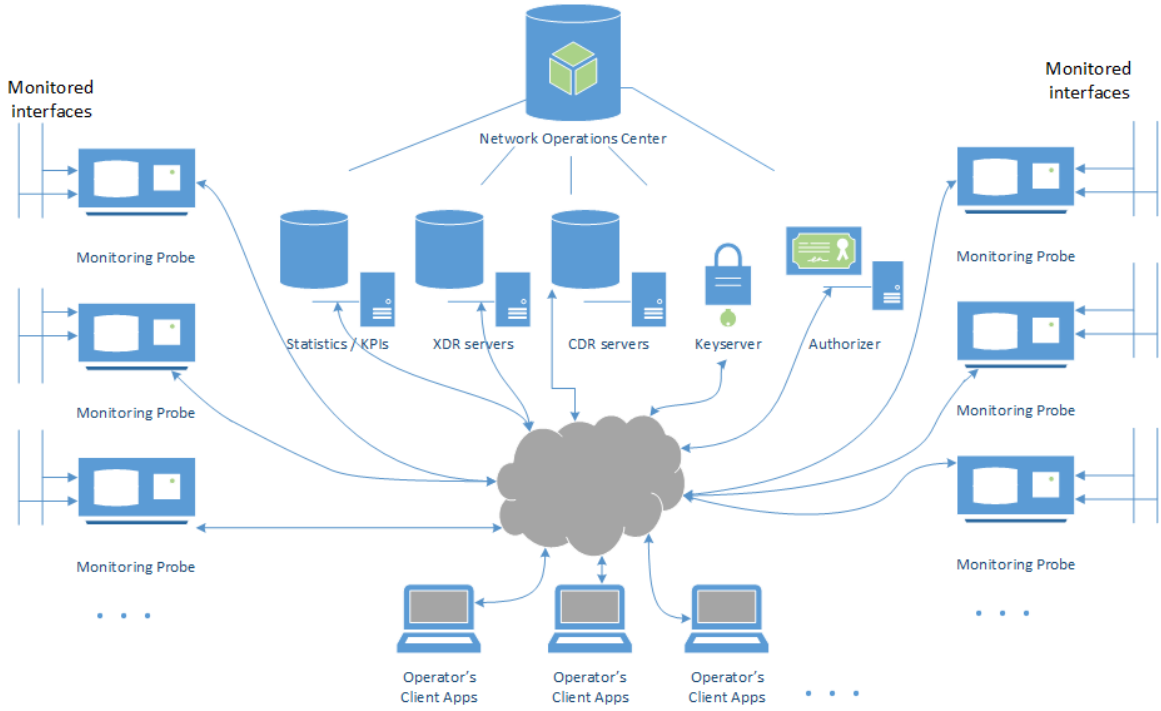

Fig. 1. A generic architecture for distributed network monitoring

II. Network Monitoring supported by FPGA-based Probes

A. The Generic Concept of Distributed Network Monitoring

The distributed network monitoring architecture depicted by Figure 1 supports local, probe-based pre-processing (time-stamping, requirement-based packet chunking, filtering criteria-based distribution) and central, deep analysis (correlation of messages and transactions, data record compilation, statistics generation), even on-the-fly. The time-based ordering and interleaving of messages are enabled by the hardware-accelerated time-stamping, providing nanosecond-range resolution with sub-microsecond precision. The information stored locally at the distributed Monitoring Probes can be accessed by client applications of the operator. Besides, the Monitoring Probes send pre-digested data to the Servers for correlation (creating e.g. Call Data Records, CDRs), as well as periodic reports containing their calculated statistics [3].

Since user data and control data are often carried over the same channels, their division requires message analysis on network- or transaction-level (e.g., IP- or TCP-level). The changing traffic patterns force the operators to look for new tools to process even the user traffic. The first step towards this is the compilation of XDRs (eXtended Data Records) based on control- and user-plane messages and transactions. These often contain message-level timestamps, as well. Based on these data, the deep traffic analysis tools provide valuable information towards business-intelligence and network optimization. Besides, all nodes can be configured to report directly to the NOC (Network Operations Center).

Operators use the network-wide, passive monitoring for fault detection, service quality assurance, and resource planning, among others [4]. Besides lossless data capture, network monitoring covers further functions, as well:

- precise time-stamping, ordering;
- compilation, search and fetch of Call Data Records (CDRs) and Extended Data Records (XDRs);
- calculation and reporting of Key Performance Indicators, KPIs;
- Call Tracing at various complexity levels;
- bit-wise message decoding for protocol analysis; etc.

All these functions are present in the network monitoring practice, since beside user-level data analysis, network analysis is important from connection-level to application-level, as well.

System elements of the described generic architecture can be implemented in many ways. In the SGA-7N system – which serves as the base implementation for the presented solution – monitoring probes of the presented system are called "Monitors". These consist of three main building blocks: a high performance Field Programmable Gate Array (FPGA)-based custom hardware platform, a firmware dedicated for network monitoring, and the probe software [5].

B. FPGA-based packet processing

There are many features that make FPGAs useful in packet processing tasks [6]. The main concept itself allows *parallel processing* of the input data. Different, simultaneous tasks can be carried out at each clock cycle on the same data,

which in this case is the packet header [7], [8]. Besides, the *input word length* is much greater for FPGAs (getting 90 bytes) than for modern CPUs (64 bits). Furthermore, FPGA are set up in hardware-defined languages, and they are indeed *reconfigurable* hardware: their internal wiring can be changed within milliseconds. These features enable FPGA-based hardware platforms to become high performance networking devices, e.g., network monitors, switches, routers, firewalls or intrusion detection systems [9]. Nevertheless, as a network monitoring system, it supports distributed and lossless packet level monitoring of Ethernet links for 1 or 10 Gbps.

Beside providing sufficient resources for switching and routing at 1 or 10 Gbps, the design of SGA-GPLANAR [10] and SGA-10GED [11] used in SGA-7N includes some special, network monitoring-related requirements, namely

– lossless packet capture,
– 64-bit time-stamping with sub-microsecond resolution,
– header-only capture: configurable depth of decoding,
– on-the-fly packet parsing by hardware [12],
– parameterized packet/flow generator for mass testing [13],[14].

Various applications then require other supported functionalities. As an example, the high-speed monitoring application [15] consists of the following sub-modules:

– time-stamping every frame upon reception;
– packet decoding from layer 2 up to the application layer;
– packet filtering with a reconfigurable rule-set to decide what we do with a given packet;
– packet chunking: packets can be truncated depending on the matching rule;
– packet distribution: to distribute packets by different criteria: IP flows, fragment steering, steering based on mobile core network parameters, etc.;
– packet encapsulation: monitoring information is stored in a specified header format.

These features and capabilities make the FPGA a suitable enabler of hardware acceleration within the Monitors.

III. Challenges and Requirements in Detail

For a distributed monitoring solution described in the previous sections, there is a strong requirement for having a *monotonic clock*. Otherwise, packet reordering would happen even with a single monitoring node (changing its clock) – and this is not feasible, since traffic analysis is heavily dependent upon packet timestamps. As a consequence, the need for monotonic system time is inherent.

Another challenge comes from the fact that a distributed monitoring system has its components geographically separated from each other, therefore the clock frequency and the time information of the clocks of the nodes have to be frequency- and phase-synchronized to each other with some given threshold. This problem has many solutions, e.g., using GPS based synchronization systems [16]. Although technically it can work well [17], as a drawback, this requires additional installation expenditures on an indoor site that has no installed antenna system to carry the GPS signal inside the building and could also result in extensive cabling work. A convenient alternative is to use network time synchronization that utilizes the telecommunication network for exchanging packets as per a designated protocol to achieve frequency and phase synchronization. Examples for this are Network Time Protocol (NTP) [1] and Precision Time Protocol (PTP) [2].

When speaking about time synchronization, the following properties describe a clock – which are in-line with the generic definition of clock properties [18]:

- accuracy – *i.e.* how good is the time information compared to some reference
- precision – *i.e.* how precise is a tick of the clock compared to some reference
- stability – *i.e.* how does the clock frequency change e.g., over time or based on external temperature changes etc.

The biggest challenge of all – as usual – is to adapt to the existing monitoring framework described in II with minimal modifications to the existing solution, while satisfying all the precision and accuracy related requirements. As mentioned before, the platform for proof-of-concept is the SGA-7N monitoring system, which utilizes FPGA-based monitoring cards. These are capable of capturing on high-speed network interfaces – with fine-grained time-stamping capabilities –, and they have their own, existing time-keeping facilities.

In order to tackle all the above mentioned issues with a solution fitting into the network monitoring architecture, we suggested to create a new FPGA-based card that implements these functions:

- network time synchronization,
- local time synchronization,
- interfacing with the existing nodes – OAMP functions.

The following sections describe this solution, and show its feasibility in the running monitoring system.

IV. Architecture of the Distributed Time Synchronized Monitoring System

A. Generic concept

For providing easy adaptation into the existing system, and also taking into account FPGA resource usage, a hybrid solution has been designed. This solution implements network time synchronization in a standalone card that distributes the digital timing information over a dedicated control bus, as illustrated by Figure 2.

The synchronization framework provides a platform-independent agent that can be integrated into the existing FPGA cards' top level VHDL (VHSIC Hardware Description Language, [19]) modules, and is used through a well-defined and portable interface.

The agent itself has low complexity, and as a result, the solution does not waste CLB (Configurable Logic Block) resources – as if the whole network synchronization stack were instantiated N times on all monitoring node cards. Furthermore, this results in better internal synchronization compared to the replicated stacks, since those can have skew to each other (within the boundaries), as specified by their protocol.

As shown by Figure 2, each node has its own network synchronization function, therefore the accuracy and precision

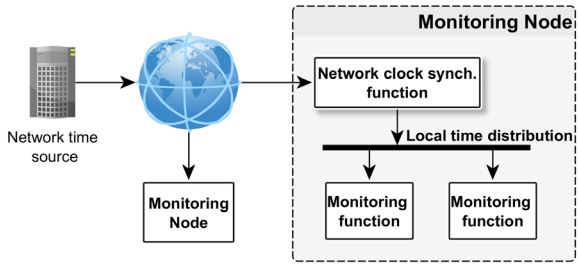

Fig. 2. Fitting the time synchronization function into the generic, distributed network monitoring concept

between two monitoring nodes can be guaranteed only to an extent that the utilized time synchronization protocol provides. Due to the uncompensated delay of routers, switches and transmission paths, this is in the magnitude of milliseconds of a software implementation of NTP. This precision, can be increased by using FPGAs for hardware acceleration. Depending on the PTP version and the underlying network capabilities, this can fall into the magnitude of nanoseconds.

The main idea of the solution is to install a local time-distribution bus between the nodes within a site. This allows us to achieve nanosecond-range synchronicity, as there is less perturbation between the hardware implementations of the transmitting and receiving ends – no OS scheduler, no network etc. Moreover, frequency synchronization can also be easily achieved by implementing a synchronous bus – i.e., transmitting the clock signal along with the data.

B. External time synch. subsystem design and implementation

When selecting the candidate for implementing the external time synchronization function, three protocols were considered:

- Network Time Protocol (NTP) [1],
- Precision Time Protocol v1 (PTPv1) [20],
- Precision Time Protocol v2 (PTPv2) [2].

In order to achieve the best synchronization between PTPv2 clocks, the protocol requires PTPv2-enabled switches/routers throughout the network. These do the bookkeeping of the processing delay values in the synchronization packets as they traverse through the network. Without this feature, the achievable synchronicity in a multi-hop network is around the same as by using PTPv1.

Since PTPv2 is not widely available in current networks, we concluded in either selecting NTP or PTPv1, due to their simplicity. PTPv1 has way more modes of operation when compared to NTP. Still, these two protocols are operating based on semantically the same principle when determining the round trip time and offset compared to a reference clock entity. Although there are significant differences originated from their packet structure, the time-stamp format and also the epoch that could result in more complex implementation if PTPv1 would be chosen. Still, the NTP time-stamp format includes a 32-bit unsigned seconds field spanning 136 years and a 32-bit fraction field resolving 232 picoseconds the prime epoch, or base date of era 0, is 0 h 1 January 1900 UTC – i.e., when all bits are zero.

Based on the requirements, the above considerations, and the Occam principle, the design decision led to selecting *NTP protocol to be used* for synchronizing the FPGA-based monitoring cards through a card that is responsible for implementing the external and internal (see Section IV-C) time synchronized function called SGA-Clock.

Each FPGA-based packet processing and networking protocol implementation has its own complexity. There are several readily available implementations that can be used for packet processing in FPGAs with some limited flexibility when it comes to interconnecting it with other modules. The one that has been used for the current implementation is a flexible solution for Protocol Implementations within FPGAs. The solution detailed in [21] provides a generic framework in VHSIC Hardware Description Language (VHDL) that enables rapid prototyping of networking protocols. Among many other things it provides the following main features:

- supports protocol module interconnection via layering;
- handles reception and transmission of Protocol Data Units (PDUs) with queuing;
- provides a high level interface for separating and combining Protocol Control Information (PCI) and Service Data Unite (SDU), forwarding, pausing or dropping SDUs;
- provides a unified way to handle Interface Control Information (ICI), SDU, and PDU events (e.g., error signalling) [22];
- adds support of auxiliary information that travels along with messages
- provides components for common tasks recurring during implementing networking protocols (de/serialization, arbitration etc.).

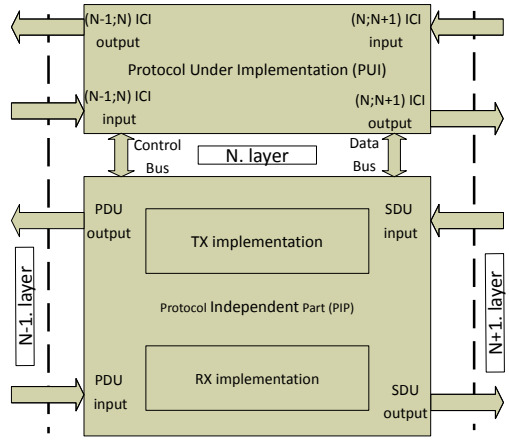

Fig. 3. Fundamental building block of the FPGA networking framework used for the Protocol Implementation

The framework's basic building block (shown by Figure 3) was used for implementing a pure FPGA-based UDP/IP protocol stack with ARP [23] on top of 802.3 Ethernet. It provides a platform with deterministic timing for the likewise FPGA-based implementation of NTP. For each of these protocols the

corresponding protocol-specific parts have been described in VHDL, using the generic framework [21].

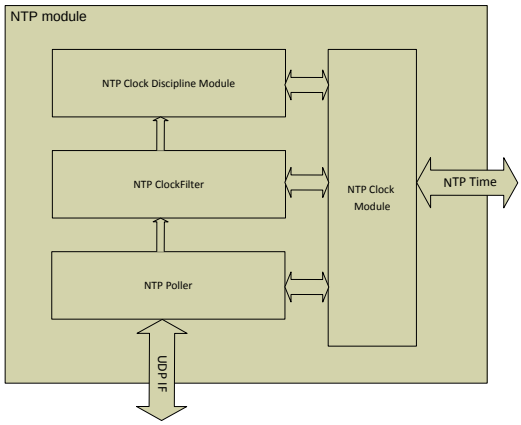

Fig. 4. NTP module block diagram of components

The internal structure of the NTP module is shown by Figure 4. The NTP Poller component is responsible for the NTP packet transmission and reception, and implementing the On-Wire protocol for determining the offset – based on the packet messages. The packet-handling part is also implemented through the Protocol Implementations framework. The NTP ClockFilter component is there to regulate the offset values presented by the poller by ordering the results based on delay, updating internal state variables, calculating jitter, and suppressing spikes based on jitter and last successful test time. If the offset data got passed the filter stage, it gets forwarded for further processing by the NTP Discipline module.

The NTP Discipline module controls the clock module – by adjusting the time increment – based on the filtered offset data. The NTP clock module provides an interface for controlling the time increment that itself is added to the clock register in each system clock cycle – thus implementing the clock functionality. The time information is fed back to each module as illustrated on Figure 4. This chain of modules with the feedback is another realization of a closed loop control chain described in the following section.

C. Internal time synch. subsystem design and implementation

Since time-stamping is done by the monitoring interface cards, the time synchronization information has to be spread around all interface cards of all monitoring units within the site. This time synchronization is an internal matter of the monitoring system. The relationship between "external" and "internal" time synchronization is shown by Figure 5.

The internal time information synchronization function is responsible for having all clocks in all monitoring functions to be completely synchronized *within a monitoring node*. Since this is an internal component, the amount of perturbation that potentially affects this subsystem is considered minimal compared to the external time synchronization subsystem.

The elements of this subsystem are:

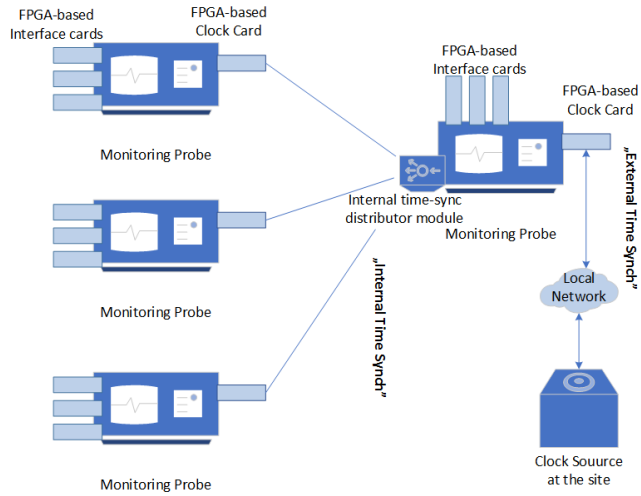

Fig. 5. Time synchronization within a monitoring site – methods for external and internal subsystems differ to allow high precision and accuracy in time-stamping

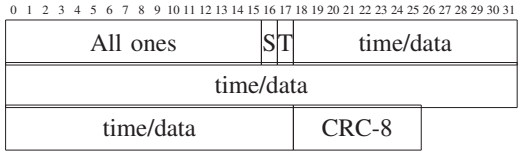

Fig. 6. High-Speed Time-stamp Interface frame format

- digital bus that is able to transmit time and status information;
- a driver module of that bus that resides in the Network clock synchronization function;
- receiver modules attached to that bus performing local time synchronization.

Internally to each monitoring probe, all FPGA boards that implement a monitoring function can operate from different power supply units. As a consequence, ground level isolation is necessary over the bus. For reducing the physical layer complexity, a point-to-point bus system has been designed. In order to be able to maximize the number of clients connected the bus, it utilizes an asynchronous serial communication using 2 wires that provides uni-directional communication – with this system bi-directional communication would require 4 wires. The communication protocol executed by the driver module (the internal time synch. distribution module in Figure 5) multiplexes arbitrary data units and the time information over the bus into frames – equipped with error detection code – in an alternating pattern. That results in periodic transmission of valid time information.

The parameters of the physical signalling are:

- LVCMOS33 (Low Voltage CMOS 3.3) levels for representing logical values;
- asymmetric signal transmission;
- 15.625 MHz clock frequency with 4x oversampling;
- NRZ line coding.

The frame format used on the bus is shown in Figure 6. The

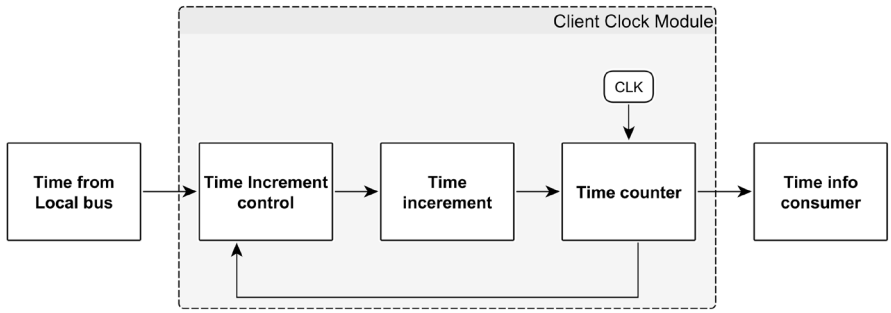

Fig. 7. Internal clock module diagram

frame starts with an all 1's preamble, and it is followed by a start bit with value 0. The type field is used to differentiate the payload types. When T=1 it indicates that the payload is time information otherwise it is data – hence an overlay data communication protocol can be used on this data channel. The time format is in line with the external time synchronization, i.e., it uses the NTP time format for representing the time information. For detecting transmission errors on the bus, a CRC-8 value is calculated for the *'Type'* and *'Payload'* fields and appended to the frame that is checked on frame reception for detecting transmission errors.

$$T_{frame} = N_{bits} \times T_{bit} \tag{1}$$

and

$$T_{bit} = 1/f_{signalling} \tag{2}$$

Where T_{bit} is the bit time, $f_{signalling}$ is the signalling frequency on the internal bus, and N_{bits} is the number of bits in the frame. Calculating (1) and (2) with the above given parameters, the frame time is 90/15.625 MHz = 5.76 µs. Since every other frame carries time information, the clock on the receiver side can be disciplined/controlled on a 11.52 µs basis. Under such short period of time even low quality, non-temperature controlled crystal oscillators have negligible drift. As a result, this update period is adequate for the network monitoring use case.

The receiver module (i.e., in the FPGA-based Clock Card in Figure 5) de-multiplexes the data and the time information from the payload of the received frame. It also verifies that the received frame's CRC-8 value matches the calculated one. If no errors were detected then it feeds the time information into a module that performs time synchronization executing the pseudo-code in algorithm 1

The client clock module – as shown in Figure 7 – is incrementing a clock counter with an increment value – corresponding to the nominal clock frequency in the internal time representation – in each system clock period. The clock module frequency can be adjusted through modifying the time increment itself. The $Delay_{static}$ constant can be measured for a given configuration and adjusted accordingly. The algorithm is illustrated by a sample waveform of the master and slave entity in Figure 8. Informally if the skew is less than the desired precision under one synchronization period – i.e. when the valid time is transmitted by the master entity – then the phase of time progresses in sync on the two entities.

Algorithm 1 Receiver local time disciple algorithm

$Increment \Leftarrow 1/f_{clk}$
$Delay_{static} \Leftarrow x$
$T_{local} \Leftarrow 0$
for Each rising edge of system clock **do**
  if Received valid time-stamp from Master **then**
    $T_{local} \Leftarrow T_{recv} + Increment + Delay_{static}$
  else
    $T_{local} \Leftarrow T_{local} + Increment$
  end if
end for

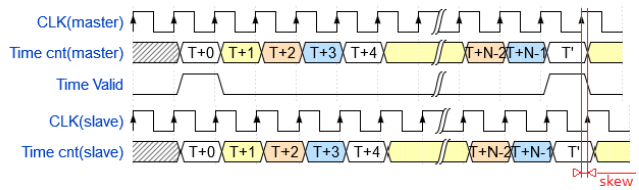

Fig. 8. Illustration of timing on the internal synchronization bus

To concede that this system can have nanosecond synchronization let us execute the algorithm: let T_n be the nth time point where synchronization occurs between the master and the slave entity. At time point T_n when the client receives a time-stamp the local clock will be in sync with the master clock – given that the $Delay_{static}$ constant was determined correctly. De-synchronization arises due to errors in the master and client clock oscillator frequency. To ensure that the desired level of synchronization is reached it has to be shown that the master and client clock would not diverge more than one nanoseconds under time interval (T_n, T_{n+1}) – since by definition at T_{n+1} the clocks will be in sync again and this process is periodic. Given a worst case calculation the following equation must hold true:

$$\begin{aligned} \left|\frac{1}{(1+\epsilon)f_{clk}}N - \frac{1}{(1-\epsilon)f_{clk}}N\right| &\lesssim 1ns \\ \frac{N}{f_{clk}}\left|\frac{-2\epsilon}{1-\epsilon^2}\right| &\lesssim 1ns \\ T_{ts}\left|\frac{-2\epsilon}{1-\epsilon^2}\right| &\lesssim 1ns, where \\ N &= \frac{T_{ts}}{T_{clk}} \end{aligned} \tag{3}$$

In equation 3 ϵ stands for the precision of the oscillator, f_{clk} is the system clock frequency and T_{ts} is the time-stamp frame time on the internal synchronization bus. In theory equation 3 can be satisfied for arbitrary ϵ if T_{ts} can be adjusted freely. Substituting parameters from the concrete implementation with $T_{ts} = 2T_{frame}$ = 11.52 µs and ϵ = 50 ppm results in $1.15ns \lesssim 1ns$ which is approximately satisfying.

It is important to note that this accuracy and precision is only achieved over the *internal* time synchronization bus. If the external subsystem – that is completely orthogonal to the internal subsystem – synchronizes to its reference with µs accuracy then this results in the same accuracy for the monitoring probe vs. external reference relation. Even though the synchronicity will be still at the ns level in the monitoring probe vs. monitoring probe relation inside the same monitoring node driven by the same master.

D. Implementation

The realized system with all internal components is shown by Figure 9, where the external time synchronization – as presented in Section IV-B – is done by the SGA Clock card – visible in the bottom right part. Similarly, the internal time synchronization – described in Section IV-C – is performed over the local bus with agent modules. These modules run in all the FPGA-based monitoring cards acting as slaves at the high-speed time-stamp interfaces; all are driven by the SGA Clock card acting as a master.

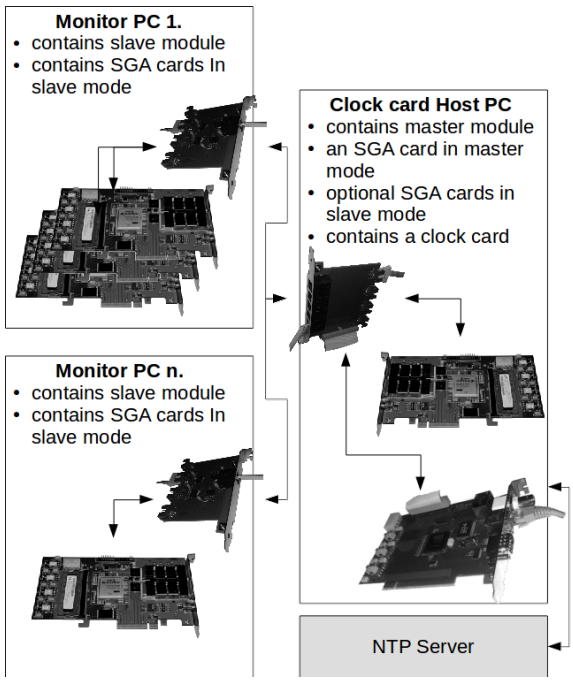

Fig. 9. The realized system with all internal components

V. VERIFICATION & RESULTS

There has been extensive testing and measurements carried out for verifying the solution. In order to analyze the degree of synchronization to the master NTP clock, a packet capturer was installed on the Ethernet segment at which the FPGA implementation of the NTP slave was connected. The NTP packets used for synchronization were captured bidirectionally. This packet capture then was filtered for those NTP packets that had all 4 timestamps used in the On-Wire protocol to calculate the offset from the reference clock value. Our dedicated post-processing utility then extracted the offset information along with the time elapsed from the start of measurement – which is determined by the first NTP packet present in the packet capture.

A sample packet capture is shown by Figure 10. The statistical parameters – like the clock drift and real offset – of the device was determined by fitting a linear curve on the offset values. There were various measurements carried out – for the actual measurement presented in this paper, the capture was taken for approximately 3 hours. The results and the fitted curve plot can be seen on Figure 11.

The curve fitted on this measurement shows that there was a fix 14.52 µs offset compared to the reference clock. As presented in Section III having a precise and stable clock – with known offset – is as good as having an accurate one. Besides, the first order stability of the device is $-0.7\,1/\mathrm{ns}$. This precision and stability are considered adequate for satisfying the requirements of the external time synchronization part.

As for the internal synchronization part, it is by design has accuracy and precision in the magnitude of 1 ns – see Section IV-C for details.

The $\sim$1 ns accuracy over the internal synchronization bus satisfies the criteria of any monitoring system with 1, 10 or even 100 Gbit/s Ethernet, since packet inter-arrival time even of the latter case is 6.72ns [24] – almost one magnitude greater than the theoretical accuracy of the presented, implemented and verified time synchronization method.

As a real-life verification, the above described time-synchronization system has been put into operation at Magyar Telekom. The system hardware (with its FPGA firmware) has been installed beside the SGA-7N network monitoring system, and showed the expected result. The system provides accurate time information to the monitoring cards ever since, and it is planned to be expanded for covering all related monitoring cards, network-wide.

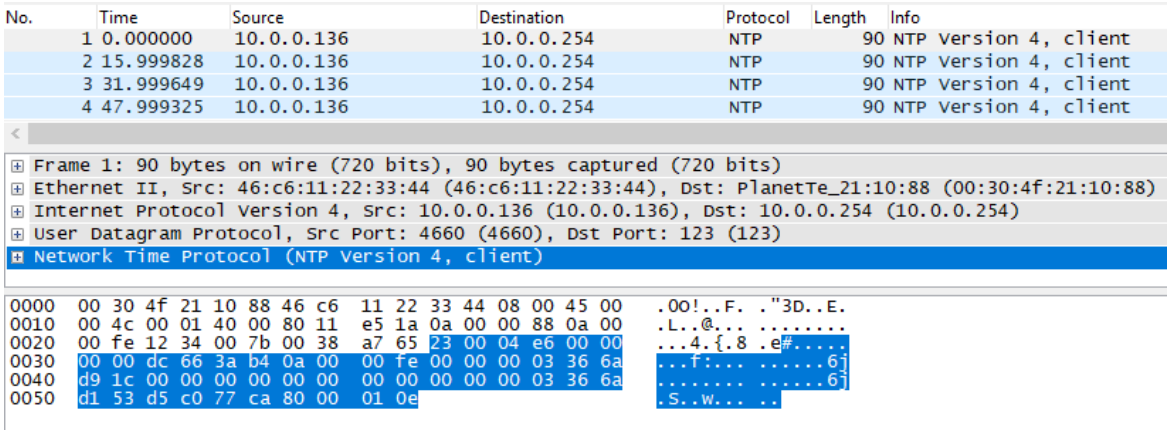

Fig. 10. Measurement result – a typical example of an NTP packet containing four timestamps

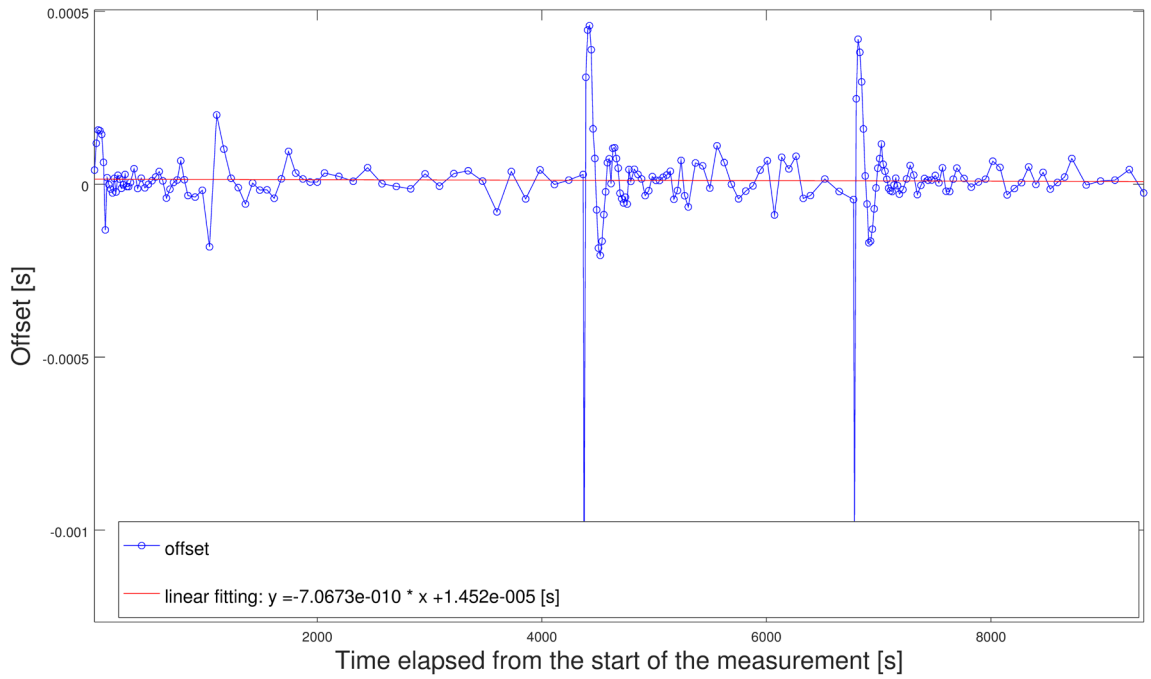

Fig. 11. Measurement result – plot of derived offset-measurement data

VI. CONCLUSION

In this paper, we introduced a general time synchronization solution for a high performance, lossless network monitoring system called SGA-7N that is based on a reconfigurable architecture. The probes of the system are called "Monitors", which consists of three main building blocks: a high performance Field Programmable Gate Array (FPGA)-based custom hardware platform, a firmware dedicated for network monitoring, and the probe software. The reconfigurable property of the FPGA chip enables to turn the Monitor hardware platform into a high performance networking device – among others, a network monitoring probe. Beside supporting distributed and lossless packet level monitoring of Ethernet links for 1 or 10 Gbps of the described system, the FPGA serves as the base platform of the time synchronization solution for the interface cards of the Monitors.

Time synchronization of the network monitoring nodes are crucial, since the analysis depends highly on the proper message sequence, which is determined mainly by the timestamps.

First, each monitoring site has to have a reference clock that is synchronized with other reference clocks at other sites. Naturally, the monitoring system has to be synchronized to the reference clock available at the physical site. In this paper we call it external time synchronization, and it is solved by an FPGA-based, NTP implementation that use data filtering and has a clock discipline module in order to output monotonous clock information. This avoids timestamps

jumping backwards, or jumping forward too much within one step, hence the clock if the monitoring system becomes well-regulated. Each interface at the monitoring node has to get synchronized with this clock information. In this paper we call it internal time synchronization, and it is implemented through a proprietary time-synchronization protocol. Its sender, (or master) part works in a distributor-card residing at the main reference clock machine of the monitoring system, whereas the receiver (or slave) parts are realized within the FPGA of the monitoring cards.

As presented in the paper, the overall system shows sub-nanosecond accuracy and stability, meeting the requirements of 10 Gbps, or even 100 Gbps Ethernet-based packet monitoring. The presented solution is already installed in the network-wide, real-life monitoring at Magyar Telekom.

REFERENCES

[1] D. L. Mills, *Computer Network Time Synchronization: The Network Time Protocol*. Boca Raton, FL: CRC Press, 2006.

[2] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (PTPv2)*, IEEE Standard 1588-2008.

[3] P. Tatai, G. Marosi, and L. Osvath, "A flexible approach to mobile telephone traffic mass measurement and analysis," in *18th IEEE Instrumentation and Measurement Technology Conference*, Budapest, Hungary, May 2001.

[4] D. Kozma, G. Soos, and P. Varga, "Supporting LTE Network and Service Management through Session Data Record Analysis," *Infocommunications Journal*, vol. 71, no. 2, pp. 11–16, June 2016.

[5] AITIA, "SGA-NETMON The GSM/GPRS/UMTS/LTE Network Monitoring System," White Paper, 2012. [Online]. Available: http://sga.aitia.ai/pdfs/SGA-NetMon.pdf

[6] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA - An Open Platform for Gigabit-rate Network Switching and Routing," in *IEEE MSE*, San Diego, CA, US, 2007.

[7] N. Possley, "Traffic Management in Xilinx FPGAs," White Paper, 2006. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp244.pdf

[8] W. Jiang and V. K. Prasanna, "Scalable Packet Classification on FPGA," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 20, no. 9, pp. 1668–1680, August 2011.

[9] P. Orosz, T. Tothfalusi, and P. Varga, "C-GEP: Adaptive Network Management with Reconfigurable Hardware," in *14th IEEE International Symposium on Integrated Management (IM)*, Ottawa, Canada, 2015.

[10] *SGA-GPlanar product description*, Aitia International Inc., accessed: 2017-12-30. [Online]. Available: http://www.fpganetworking.com/gplanar/

[11] *SGA-10GED product description*, Aitia International Inc., accessed: 2017-12-30. [Online]. Available: http://www.fpganetworking.com/10ged/

[12] V. Pus, L. Kekely, and J. Korenek, "Low-Latency Modular Packet Header Parser for FPGA," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Austin, TX, USA, October 2012.

[13] P. Olaszi, "Complex Load Testing of Mobile PS and CS Core," EuroNOG 2012, September 2012. [Online]. Available: http://www.data.proidea.org.pl/euronog/2edycja/materials/Peter_Olaszi-Complex_Load_Testing_of_Mobile_PS_and_CS_Core.pdf

[14] G. Soos and P. Varga, "Use Cases for LTE Core Network Mass Testing," in *5th Mesterproba*, Budapest, Hungary, 2016.

[15] P. Varga, L. Kovacs, T. Tothfalusi, and P. Orosz, "C-GEP: 100 Gbit/s Capable, FPGA-based, Reconfigurable Networking Equipment," in *16th IEEE International Conference on High Performance Switching and Routing (HPSR)*, Budapest, Hungary, 2015.

[16] B. Sterzbach, "GPS-based Clock Synchronization in a Mobile, Distributed Real-Time System," *Real-Time Systems*, vol. 12, no. 1, pp. 63—-75, January 1997.

[17] J. J. Garnica, V. Moreno, I. Gonzalez, S. Lopez-Buedo, F. J. Gomez-Arribas, and J. Aracil, "ARGOS: A GPS Time-Synchronized Network Interface Card based on NetFPGA," in *2nd North American NetFPGA Developers Workshop*, Stanford, CA, USA, August 2010.

[18] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, October 1991.

[19] *IEEE Standard VHDL Language Reference Manual*, IEEE Standard 1076-2008.

[20] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Standard 1588-2002.

[21] F. N. Janky, "A Flexible Architecture for Protocol Implementations within FPGAs," in *25th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, 2017.

[22] *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*, ISO/IEC Standard 7498-1:1994(E).

[23] *An Ethernet Address Resolution Protocol*, IETF Internet Standard RFC 826.

[24] P. Mooney, "40/100-Gigabit Ethernet: Watching the Clock," White Paper, 2009. [Online]. Available: http://www.lightwaveonline.com/articles/print/volume-26/issue-10/applications/40100-gigabit-ethernet-watching-the-clock.html

Ferenc Nandor Janky received the MSc degree in Electrical Engineering from BME, Budapest, Hungary, in 2013. He gained experienced while working for various telecommunication companies including Vodafone, AITIA International Inc. and Ericsson. His main areas of interest are network protocols, FPGA programming and software development. Ferenc is currently working as a C++ software developer for an international corporate bank. He is a member of the SmartComLab at BME TMIT.

Pal Varga is Associate Professor at BME, Hungary, where he received his Ph.D. (2011) from. Besides, he is director in AITIA International Inc. Earlier he was working for Ericsson Hungary and Tecnomen Ireland, as software design engineer and system architect, respectively. His main research interest include network performance measurements, root cause analysis, fault localization, traffic classification, end-to-end QoS and SLA issues, as well as hardware acceleration, and Internet of Things. He is also a member of the SmartComLab at BME TMIT.