High-Speed, In-band Performance Measurement Instrumentation for Next Generation IP Networks

Dimitrios P. Pezaros, *MIEEE*
Department of Computing Science
University of Glasgow
Glasgow, UK
G12 8QQ
dp@dcs.gla.ac.uk

Konstantinos Georgopoulos
Engineering Department
Lancaster University
Lancaster, UK
LA1 4YR
k.georgopoulos@lancs.ac.uk

David Hutchison, *MIEEE*
Computing Department
Lancaster University
Lancaster, UK
LA1 4WA
dh@comp.lancs.ac.uk

Abstract. Facilitating always-on instrumentation of Internet traffic for the purposes of performance measurement is crucial in order to enable accountability of resource usage and automated network control, management and optimisation. This has proven infeasible to date due to the lack of native measurement mechanisms that can form an integral part of the network's main forwarding operation. However, Internet Protocol version 6 (IPv6) specification enables the efficient encoding and processing of optional per-packet information as a native part of the network layer, and this constitutes a strong reason for IPv6 to be adopted as the ubiquitous next generation Internet transport.

In this paper we present a very high-speed hardware implementation of in-line measurement, a truly native traffic instrumentation mechanism for the next generation Internet, which facilitates performance measurement of the actual data-carrying traffic at small timescales between two points in the network. This system is designed to operate as part of the routers' fast path and to incur an absolutely minimal impact on the network operation even while instrumenting traffic between the edges of very high capacity links. Our results show that the implementation can be easily accommodated by current FPGA technology, and real Internet traffic traces verify that the overhead incurred by instrumenting every packet over a 10 Gb/s operational backbone link carrying a typical workload is indeed negligible.

1. Introduction

Since its early days, the Internet has adopted the conceptual separation between data and control planes, following the typical paradigm of telecommunication networks design. The data plane employs a layered structure to hide complexity while enabling scalability and efficiency, and the control plane is designed to cut across the layering in order to provide visibility to all aspects of the network which must be monitored and managed [4]. In contrast to traditional communication networks however, the all-IP packet-switched Internet does not adopt any notion of end-to-end (virtual) circuits that can be individually managed and provisioned through dedicated control channels. End-to-end data and control traffic is multiplexed at the level of individual datagrams (packets) under a single best-effort delivery service, which constitutes accountability of resource usage, traffic performance evaluation and native network control non-trivial in small timescales and –in many cases– infeasible.

The next generation Internet that will constitute a global telecommunications medium, will need to provide consistently predictable levels of service which will be dynamically managed and controlled at various timescales and granularities. Accountability of resource usage will be of primary importance for the provisioning of the infrastructure, the automated short-term traffic engineering, and also the validation of perceived performance's conformance to the agreed service level specifications between providers and customers. For such purposes, existing performance measurement mechanisms are clearly inadequate since they have been engineered as distinct and add-on control plane structures that are not fully integrated with the Internet's forwarding operation, and are therefore subject to network administrative constraints [21], device's temporal load [7], and to non-identical treatment by the network nodes [11]. Pervasive instrumentation mechanisms are needed that will integrate with the network's data plane, and will directly assess the true performance of the operational traffic at small timescales, hence becoming integral part of automated closed-loop network control and management entities.

Previous work by the authors has concentrated on the design of an in-band measurement mechanism to seamlessly integrate measurement functionality with the data plane's main forwarding operation [16]. *In-line* measurement uses native network-layer structures of the next generation Internet protocol (IPv6) to

piggyback performance indicators to the actual data-carrying traffic while this is routed between two points in the network, and to therefore directly measure the true user-perceived performance in short timescales. A software-based prototype has been built to operate as part of the protocol stack of commodity operating system kernels and to measure the performance of individual end-to-end traffic microflows. Its applicability as well as its minimal impact on end-system performance have been thoroughly evaluated [18][17].

IPv6 has been designed as the successor of the current Internet Protocol (IPv4), and the main necessity urging for its adoption has thus far been the substantially extended address space it provides, which will overcome the ever increasing problem of address exhaustion in IPv4 [9]. However, IPv6 also adopts a complete new philosophy on the way optional and control information is carried within individual data packets and the way this is processed by network nodes along an end-to-end path (see section 2). This clean-slate design allows for measurement (and control) instrumentation to become an integral part of the network's main forwarding operation, and this is another strong reason for the protocol to be adopted as the fundamental transport for the next generation Internet. Even under partial IPv6 deployment, the transparency of its optional mechanisms and their ability to be deployed within network domains as well as end-to-end can make any such 'in-line' instrumentation fully interoperable with the current Internet architecture.

In this paper, we present a hardware implementation of in-line measurement that would enable the mechanism to operate at very high speeds (10 Gb/s) and hence to be deployed between the edges of high-speed network topologies (e.g. Tier-1 backbones) to provide for two-point in-network direct performance measurement as opposed to strictly end-to-end. Such deployment that can be integrated with the hardware (fast) path of core network devices (routers) opens up new possibilities not only for traffic measurement and performance evaluation, but also for network accounting, fault diagnosis, self-optimisation and resilience, to name a few. Instrumentation and measurement of the actual data-carrying traffic can take place at an always-on manner, and in multiple timescales and levels of granularity. For example, network traffic and demand matrices can be directly computed (instead of being approximated by correlating partial subsets of

passive measurement data) simply by instrumenting packets at network ingress and at varying levels of aggregation, and by subsequently examining structured fields of the packet contents at network egress.

The hardware implementation of in-line measurement quantifies the exact space and time requirements of the core instrumentation functionality, as this is instantiated by a set of distinct modules that operate at the source and destination of an (intermediate) instrumented network path, respectively. It highlights the exact places of the device's data path where the functionality can reside, and how it can be seamlessly integrated with the appropriate hardware controllers.

The remainder of this paper is structured as follows. Section 2 outlines the in-line measurement technique and how it can operate between two nodes over an IPv6 internetwork. Section 3 describes the particular hardware implementation and highlights the details of the various system components and their interconnections. Section 4 performs a detailed operational analysis by evaluating the worst-case overhead incurred by the inherent properties of piggybacking additional data on existing structures. Section 5 analyses the actual overhead such a hardware instrumentation system would have incurred if it was deployed over an operational high-speed network link based on recent traffic traces, and it shows that such overhead would be minimal based on actual network loads and provisioning. Finally, Section 6 concludes the paper.

2. Background

The common root behind the shortcomings of current active and passive measurement mechanisms lies in network measurement being an afterthought in the initial design of the Internet and its core components [5]. Consequently, their operation adds overhead, and they are judged according to the "do no harm" principle with respect to router performance degradation and the generation of –additional– measurement traffic. In addition, the major requirement for interoperability and backward compatibility governing all Internet mechanisms, poses great restrictions in the design of clean-slate measurement systems.

Active measurement mechanisms which usually abstract network internals by providing a measure of how traffic is routed between two points in the network, suffer from the standardised strict protocol layering and the consequential hop-by-hop processing. The existing TCP/IP stack does not offer the flexibility of

implementing pervasive measurement structures within its layers. At the same time, network nodes use an optimised fast path implementing all or part of the data plane in hardware using Application-Specific Integrated Circuits (ASIC)s. Any optional or non-standard processing has to go through the control processor (slow path) and therefore measurement traffic requiring per-hop processing is bound to be handled differently within the forwarding engines, and henceforth exhibit overall different properties than data traffic. Single-point *passive* measurement systems suffer from their inherent promiscuous operation and the immense amount of traffic they need to process. Even when operating completely independently from the network nodes using additional dedicated hardware, they do not perform a direct measurement; rather they monitor operational traffic and hence conclusive measurement can only be conducted offline, after significant amounts of data have been correlated and synthesised [8].

IPv6 has adopted a clean-slate design and apart from the extended address space, it also offers the opportunity to integrate traditional control-plane mechanisms with the main forwarding operation. This is due to the complete redesign of the encoding of optional information in the ubiquitous network-layer header, which allows for a truly extensible set of options to be implemented as a native part of the protocol stack. The protocol specification has standardised the concept of *extension headers* and *options* which are encoded immediately after the main IPv6 header and, apart from general formatting and alignment requirements, leaves options to be defined by programmers and engineers [6]. Furthermore, all network layer (apart from the explicit *hop-by-hop*) options are only processed by the node whose address appears in the destination address field of the main IPv6 header, which in most cases is the ultimate destination of a packet. This fundamental design decision enables end-to-end option processing, and eliminates the cumbersome compulsory hop-by-hop processing of all options which was the case for IPv4. In addition, the equivalent of IPv4's loose source routing is implemented using a different algorithm in IPv6, in order to avoid additional per-packet processing at every hop [6].

We have exploited native features and the option processing paradigm of IPv6 to design an in-band, direct performance measurement mechanism for the next generation Internet which will operate on the actual traffic and will provide for low-overhead traffic instrumentation. In-line measurement [16] exploits

native network-layer structures to piggyback measurement indicators to the data-carrying packets and assess the true traffic performance while it is routed between two points in the network. The ubiquitous presence of IPv6 (network layer) as the next generation Internet protocol, constitutes in-line measurement virtually applicable to any traffic type carried over the global infrastructure. In addition, selective measurement processing only at explicitly pre-identified nodes, as opposed to hop-by-hop, enables performance measurement instrumentation to operate in an always-on manner alongside the network's main forwarding functionality. We have used the IPv6 *Destination Options* extension header to define and encode two dedicated Type-Length-Value (TLV)-encoded measurement options for One-Way Delay (OWD) and One-Way Loss (OWL) measurement, respectively. Their fields, byte alignment, and their encapsulation in a destination options extension header are shown in Figure 1. The header is created and inserted to a packet before it departs from a source IPv6 node, and is distinguished by a unique *Next header* value of 60 in the immediately preceding header. It is subsequently only processed by the ultimate or explicitly pre-identified intermediate destination nodes, which can amend or simply record the measurement indicators as necessary. The rest of the nodes treat it as a higher-layer protocol and do not process it during packet forwarding.

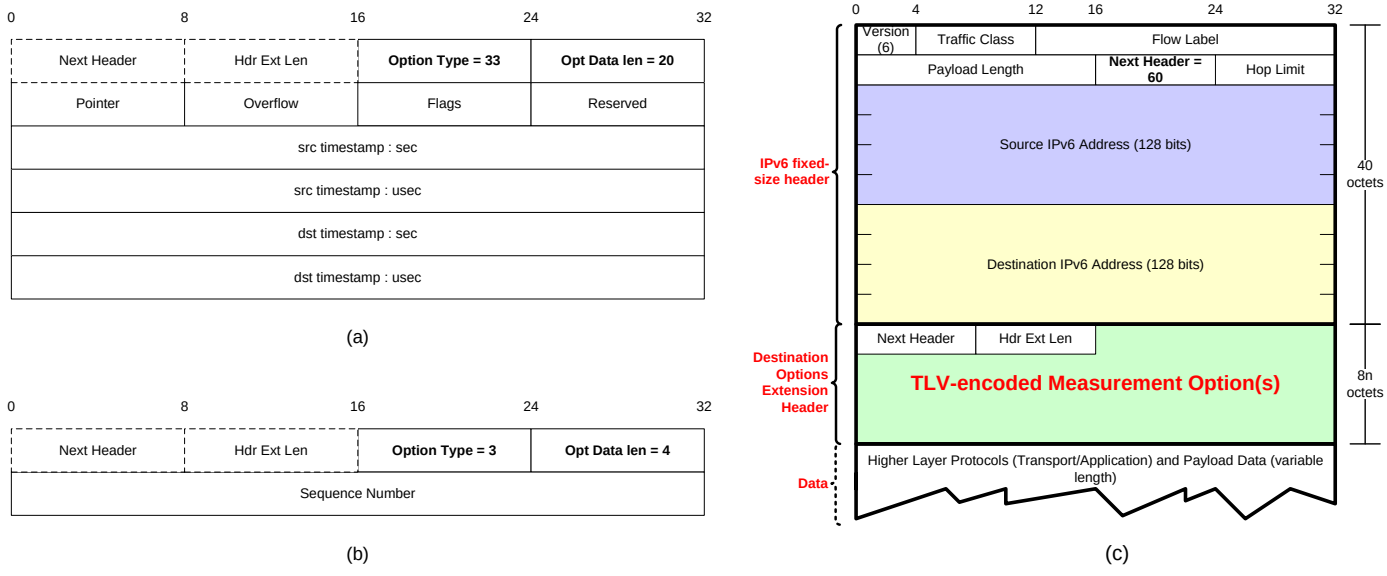

Figure 1: (a) One-Way Delay (OWD) and (b) One Way Loss (OWL) IPv6 destination options; (c) Encoding of destination options extension header within an IPv6 datagram

2.1. Two-point direct measurement

In-line measurement satisfies the fundamental pre-requisites of an efficient, always-on mechanism with ubiquitous presence over the next generation Internet architecture. It adopts a highly *modular design* that separates the traffic instrumentation (real-time) process from the rest of the measurement functionality. The minimal core measurement modules operate as a native part of the nodes' network stack, whereas higher level measurement infrastructures can be built independently. The overall framework can be extended to accommodate additional types of measurement simply by defining the corresponding options, while maintaining the same operational and processing principles. In-line measurement inherits IPv6's *selective* (option) *processing* and requires additional packet processing only at explicitly identified network nodes. Therefore, the core mechanism can be seamlessly implemented by providing additional hardware and/or software instrumentation only when and where required, leaving the rest of the network's forwarding engines unaffected. This makes in-line measurement realistically applicable over the next generation Internet since instrumentation can be built incrementally while maintaining backward compatibility with the rest of the infrastructure. Finally, the mechanism conducts a direct measurement between two network nodes, eliminating the need to correlate and synthesise data from multiple instrumented points.

The simplest instantiation of in-line measurement is over an end-to-end path, where software instrumentation modules can extend the end-systems' network stack to provide for native instrumentation at the IPv6 layer. However, in order to apply in-line measurement to instrument traffic between ingress and egress routers of ISP backbone topologies, a software-based implementation would not have been feasible, mainly for two reasons. First, it would not scale to the aggregate capacities network routers operate on unless there was serious hardware redundancy, something incompatible with an in-router implementation [19]. And second, such implementation would require traffic to be routed through the router's control processor, creating a bottleneck and resulting in instrumented traffic experiencing slower performance than the rest of the traffic [1]. Rather, measurement instrumentation should be implemented directly on a node's line cards using dedicated hardware and the per-link interface on-board processor. This approach lends itself

well to the distributed paradigm where minimal hardware subsystems are inserted where and when required, and can individually meet the requirements of very high speed backbone links.

3. Hardware Implementation of In-line Measurement Modules

Modern routers supporting multi-gigabit and terabit backplanes employ a switch-based architecture with fully distributed processors on the network interface cards. This design removes the internal interconnection bottleneck between multiple interfaces and allows per-interface heterogeneous processing elements to implement time-critical tasks directly in hardware. IP routers explicitly distinguish processing tasks related to packets forwarded through their interfaces (fast path) and those related to packets destined to the router itself (slow path). They subsequently optimise the real-time per-packet protocol processing tasks in order to achieve multi-gigabit rates, whereas they centrally process background and management tasks in software. Figure 2 shows the distinction between protocols in the forwarding path of a packet that are implemented in hardware on the network interface itself, and control protocols implemented on the CPU [1].

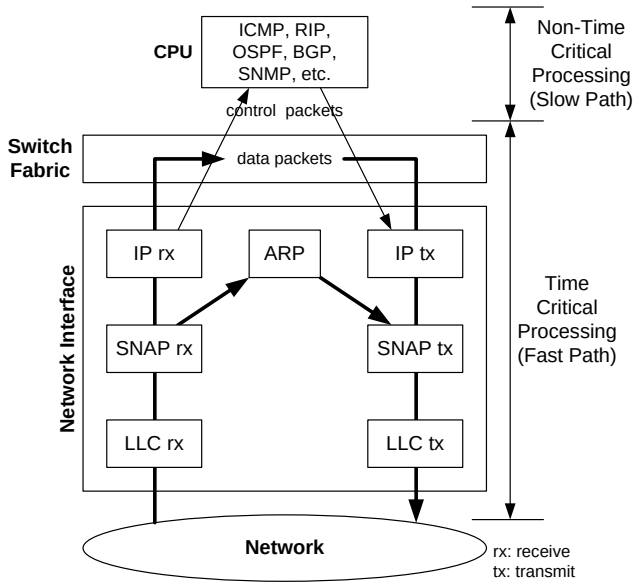

Figure 2: Example Protocol Entities in an IP router

In a typical router implementation, IPv4 options are implemented on the CPU mainly due to the way they have been standardised and engineered. The whole set of options has been monolithically designed to reside within the main IPv4 header, they need to be processed hop-by-hop by every visited node, and their implementation is mandatory by all IP nodes. The only truly optional bit is their presence within an IP

datagram. IPv4 option processing therefore requires a degree of flexibility and extensiveness which is ineffective to build in ASICs. On the contrary, IPv6 options carried within the protocol's extension headers are distinguished by a unique protocol number, and therefore nodes can implement their choice of options. Hence, an IPv6 node can accommodate (partial) option processing on its fast path by implementing a minimal subset of options directly in hardware.

In this paper, we present a method for integrating in-line measurement instrumentation into a generic IP router architecture using a reconfigurable digital circuit. We provide a Field Programmable Gate Array (FPGA) implementation which allows for very high complexity tasks to be built at significantly high clock speeds and at the same time, can form the single base design for the implementation of distinct measurement modules (e.g. conducting a different type of measurement) by being updated and re-modified in-field. FPGAs have already been employed to serve at the core of Network Interface Cards (NIC)s and recently, FPGA development platforms for research and education activities in network interface design have been deployed [20][15]. We have investigated the feasibility of using FPGAs to implement the in-line measurement functionality directly in a router's fast path, as part of the existing Media Specific Interface (MSI) or the inbound and outbound processing entities of a NIC. Our target implementation is for 10 Gigabit Ethernet interfaces which are becoming a commodity both as upstream links of access networks and at the ingress/egress Points-Of-Presence (POP)s of backbone topologies.

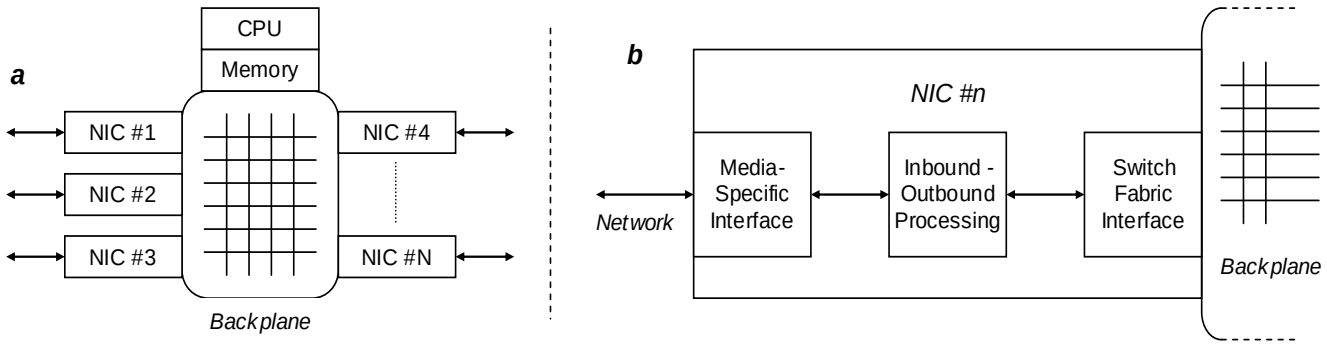

Figure 3: High-level (generic) router architecture

3.1. The system

The overall router system is mainly comprised of a number of network cards connected via a crossbar switch-backplane, as shown in Figure 3(a). CPU and Memory cores that deal with functions related to the *non-critical* data path are also shown. Our main interest, however, is focused toward the architectural hardware elements related to the *critical* data path processing (*fast path*), which mainly consists of header checking and forwarding, and is mostly implemented in special-purpose hardware [13]. Figure 3(b) provides a functional description of a NIC. It is split into three main steps: the implementation of all physical layer and Media Access Control (MAC) functions by the Media-Specific Interface (MSI), the preparation of every packet for its journey through the switch fabric by the Switch Fabric Interface (SFI), and the inbound and outbound processing of protocol functions. For 10 Gigabit Ethernet, the IEEE 802.3 MAC sublayer connects to Physical Layer entities such as 10GBASE-SR through a 10 Gigabit Media Independent Interface (XGMII) [10], as shown in Figure 4. XGMII is a parallel bus operating at high speeds which sometimes renders it difficult to implement lengths of over a few centimetres on standard Printed Circuit Board (PCB) material. Hence, the 10 Gigabit Attachment Unit Interface (XAUI) has been developed and can be optionally used to increase the operating distance of the XGMII, however, this is beyond the scope of this work. Finally, 10 Gigabit Ethernet is defined for full duplex mode of operation only and the prevalent medium of choice has been fibre optics. Specifically, the XGMII interface provides two separate 32-bit data paths, one used for data transmission (TxD<31:0>) and one used for data reception (RxD<31:0>). There are two 4-bit control signals (TxC<3:0> and RxC<3:0>) synchronous to their respective clock signals (TxClk and RxClk) that operate at 156.25 MHz ±0.01%, as shown in Figure 5. The Reconciliation Sublayer adapts the protocols of the MAC to the encodings of the 10 Gb/s PHY, converting between the MAC *serial* data stream and the *parallel* data paths of the XGMII.

The edge-to-edge in-line measurement implementation consists of the addition (at network ingress) and extraction (at network egress, for transparency) of two different extension headers, potentially on every packet. We have chosen to implement the overall functionality at the XGMII interface and prior to the Reconciliation Sublayer (RS), where packet data are received in batches of 32 bits in parallel.

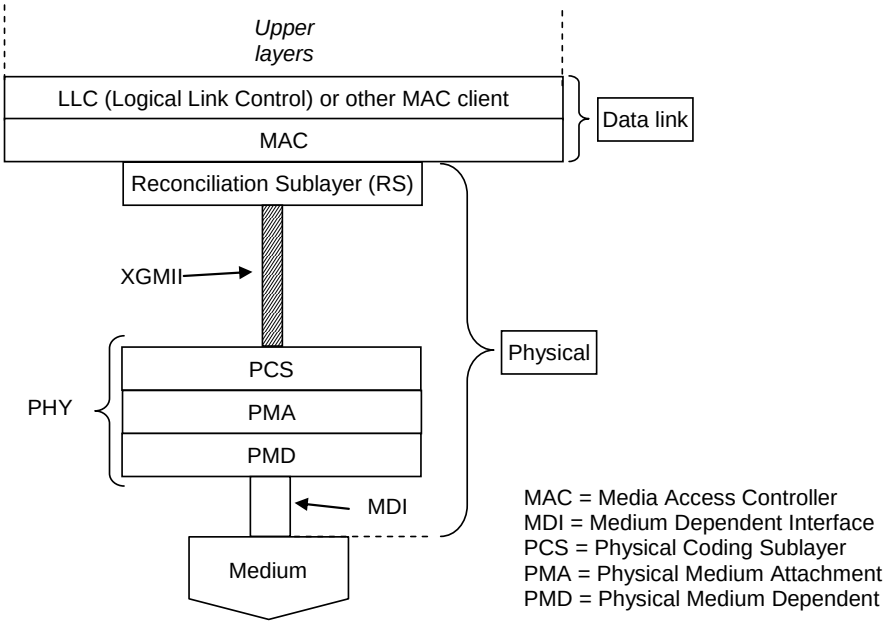

Figure 4: Architectural representation of 10 Gigabit Ethernet with XGMII [10]

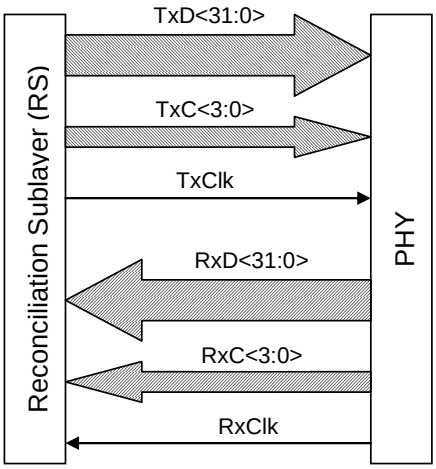

Figure 5: Full duplex operation of XGMII

This can be taken advantage of since header addition and extraction can take place at the assumed data rate of the 10-Gigabit interface rather than at a later time (after the RS layer) where packet data will have been serialised. The potential impact of introducing added functionality onto the design and structure of existing routers at a higher level of the protocol stack is suppressed, and it therefore keeps the in-line measurement functionality independent from particular router implementations. In addition, the possibility of diffusion of necessary alterations to lower layers triggered by a higher-layer design, as well as the significant design effort to ensure no conflicts between existing functions (of the admittedly more complex higher layers) and the in-line measurement implementation are being avoided.

Modern-day FPGA capabilities facilitate the implementation of a 10 Gigabit Ethernet MAC core along a XGMII interface with ease (leaving a significant amount of resources available for additional designing). Implementing both extension header addition and extraction on the XGMII parallel data streams means that packet processing occurs *before* the reconciliation layer for *inbound* packets and *after* the reconciliation layer for *outbound* packets, as shown in Figure 6. Filtering and sampling functions have also been implemented to enable selective and statistical instrumentation of (subsets of) network traffic at configurable levels of aggregation, and to henceforth balance between measurement granularity and overhead, as dictated by temporal and physical constraints of individual routers.

As an example of how the Extension Header Operations stage is structured, Figure 6 shows a block-level representation of all major stages required for the *addition* of measurement extension headers onto outgoing packets. The packet data are initially inspected to ensure they match the current filtering and sampling specifications, in which case the outbound packet is forwarded to the *Header Addition* block. At this stage, the outgoing packet is modified accordingly in order to accommodate the headers that hold the in-line measurement indicators dictated by the particular measurement implementation.

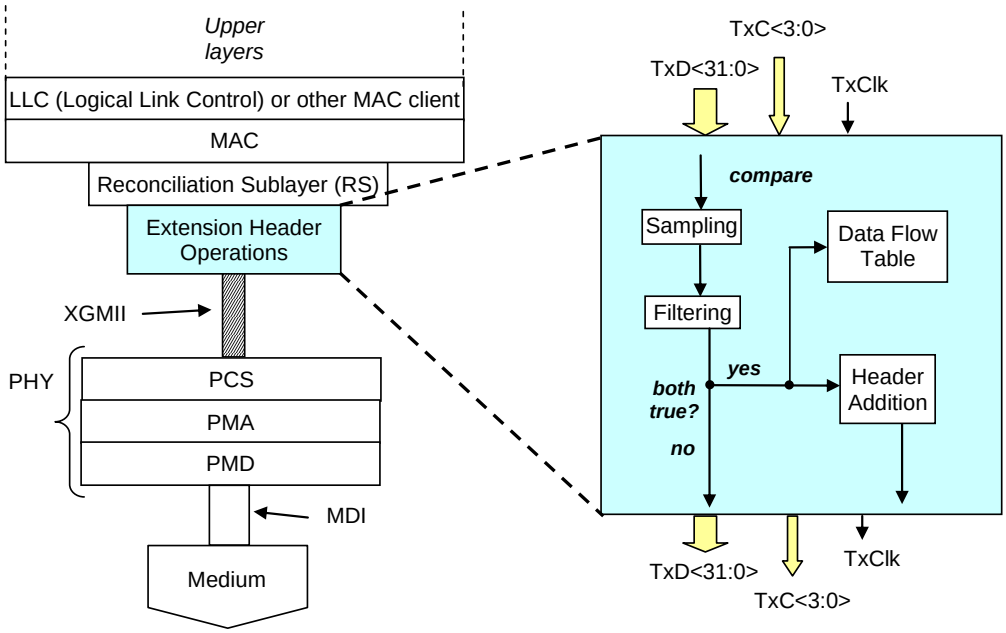

Figure 6: Measurement functions as part of the XGMII physical layer for OWL extension header addition

In case the metric of choice is stateful (like, e.g. the One-Way Loss measurement where the value inserted in each packet depends on values carried in previous packets) there is an association of each datagram with entries held in a *flow table* in order to determine the appropriate per-packet indicator. This process takes place in parallel with filtering, and the corresponding flow table entry is either created or updated (if it pre-exists) accordingly. If, however, filtering or sampling criteria are not met, the packet is forwarded unaltered.

The Header Addition stage along with the overall implementation (including data addition and extraction for both delay and loss) will be described in further detail with particular focus on the implementation challenges that have risen in realising this scheme in a particular type of FPGA.

3.2. Packet Delay and Loss Header Addition

Xilinx Virtex5 XC5VLX30T has been selected as the target implementation device and it has indeed proven to be a strong candidate for such application. The Virtex5 family of FPGAs represents a range of very efficient devices since in most cases they exhibit significantly large hardware resources. In addition, they have been fabricated with low-power consumption (65nm technology) in mind whilst ascertaining high clock speeds [23]. In fact, with careful designing, speeds up to 550 MHz can be achieved and it should be noted that the XGMII operating clock speed falls well below that value, i.e. 156.25 MHz. More importantly, Virtex5s offer Ethernet MACs readily available as Intellectual Property cores which could facilitate the employment of a FPGA on NICs. This section describes the hardware implementation for the addition of extension headers for both One-Way Delay (OWD) and One-Way Loss (OWL) in-line measurement, with a particular focus on the hardware architecture used along with information regarding its performance, such as percentage of logic cells and timing constraints. The hardware design has been developed in VHDL within the ISE 10.1 Xilinx Design Suite. One of the basic implementation principles that were followed in order to ensure the design can in fact operate at speeds greater than or equal to the XGMII recommended clock speed (i.e. 156.25 MHz) was the adoption of a state machine approach. Conditional VHDL statements have been avoided and hence the desired clock speed has easily been achieved at the cost of additional hardware resources albeit very small. Figure 7 shows the fundamental architectural implementation for the addition of measurement extension headers. It should be noted that both loss and delay in-line measurement maintain

the same broad implementation apart from subtle differences. In this figure, thick drawn lines represent parallel bit buses whereas thinner lines represent single bit values. Naturally, only the most significant transmission lines are shown here since many more are actually involved for the successful operation of the complete core. The high-level components that comprise the extension header addition process are an array of modules that carry out filtering (array of Flag Check modules), a Xilinx core for clock management (DCM), a Data Flow Table and, finally, a Header Addition module.

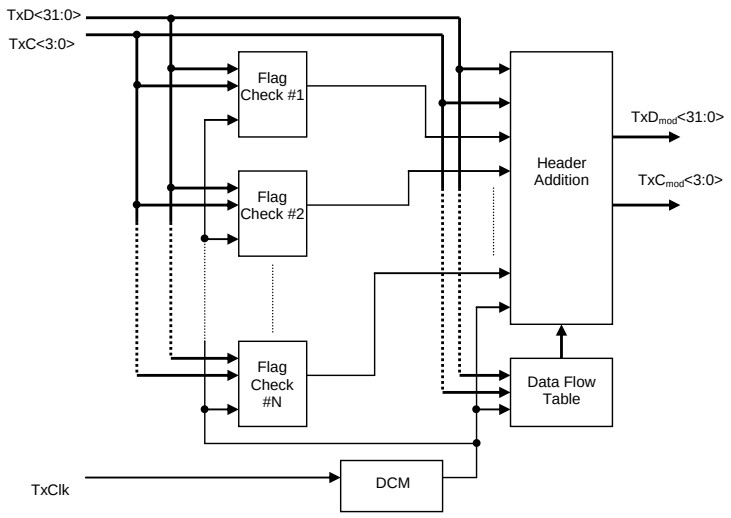

Figure 7: Block level representation of extension header addition

3.2.1. Packet Filtering

Packet filtering to select which packets will be instrumented with in-line measurement extension headers has been realised through an array of flag modules. They check for packets that belong to particular traffic flows, as these are identified by the typical five-tuple of IP(v6) source and destination addresses, transport protocol, and transport layer source and destination ports. Any of the examined fields can be set to wildcard in order to match any packet and provide for configurable filtering granularity and aggregation. For example, setting all fields to wildcards results in instrumenting all packets regardless of flow membership, whereas address prefix-based filtering can group packets into flows based on source or destination net-blocks (as opposed to distinct addresses). A total of ten independent flags have been used each one checking a different part of the packet. Each flag module compares at most 32 bits of information as they are produced in parallel by the XGMII interface every half cycle. Hence, multi-byte fields are dealt with by

multiple flag modules (e.g. four flag modules are used for IPv6 source and destination address fields). A flag is raised only if there is a match between the outgoing packet's data and the reference value, otherwise it remains low. The flags are raised in series starting from #1 and moving through to #N, as shown in Figure 8. In case all flags are raised then the *Header Addition* module can start transmitting a modified version of the original packet, i.e. one that contains the extension headers. However, as soon as a flag remains low, signalling a non-filtered packet, the Header Addition module reverts to transmitting the outgoing packet un-altered. It is important to note that the transmission of a modified packet begins after the last flag has been raised whereas an unmodified packet can begin sooner than that.

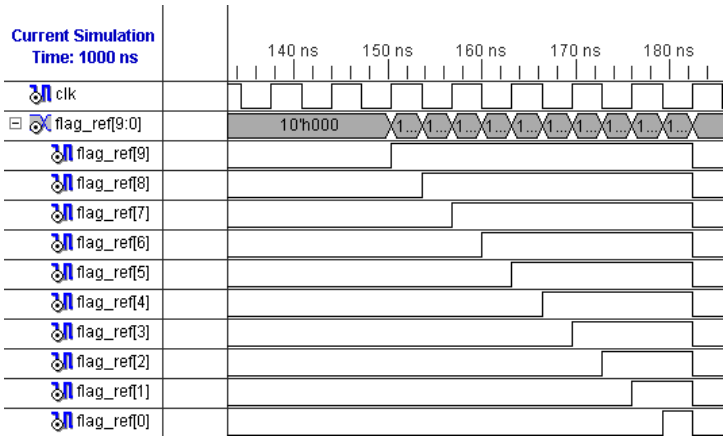

Figure 8: Sequential rise of flags for filtering

3.2.2. Clock Manager (DCM)

DCM is a soft-core developed and provided by Xilinx. It accepts a reference clock in and can output modified clock signals at different frequencies and phases. The use of the DCM in this design is that it helps emulate the Double Data Rate (DDR) principle of the XGMII. In this case, the DCM output clock is twice the frequency of the XGMII clock (TxClk). Hence, all modules operate on the rising-edge of the DCM output, which is the same as using both rising and falling edges of TxClk, as shown in Figure 9.

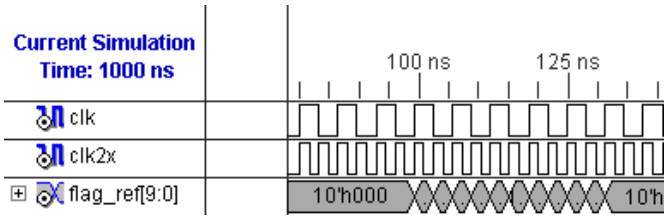

Figure 9: Clock generation for DDR operation

3.2.3. Header Addition

This is the most sophisticated component of the system since it is responsible for acknowledging incoming signals that dictate whether extension header addition should take place or not, as well as for implementing the actual functionality for extension header insertion into an outgoing packet. Figure 10 shows a block level representation of the methodology used for header addition at a minimum-sized Ethernet segment. It refers to an outgoing packet size of 64 bytes (512 bits), including link-layer headers and padding. However, this core of the hardware implementation is equally applicable to packets of different length subject to minor modifications. The packet is transmitted in batches of 32 bits in parallel which means that a 64-byte packet is transmitted in 16 sets of 32 parallel bits. Since the XGMII interface transmits data on both rising and falling edges of a clock signal, it would take 8 clock cycles to transmit a packet of 64 bytes in size. Hence, assuming that all flag signals are raised high, the packet transmission will be based on the implementation shown in Figure 10. Two groups (Group A and Group B) of 16 32-bit buffers are used in the process. *Group A* stores all outgoing data that make up a full packet, i.e. TxD is loaded onto the first group of 16 buffers. Similarly the second group (*Group B*) of buffers unloads data onto TxD_{mod}. Signal *Cntrl* marks the event where all *Group A* buffers are filled with the contents of the packet that is to be modified.

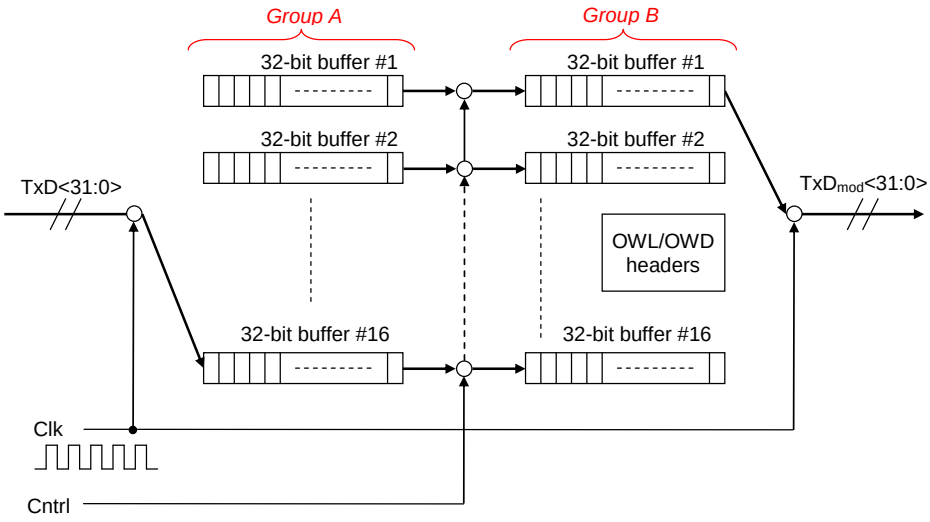

Figure 10: Schematic for Header Addition module (64-byte packet)

As soon as *Cntrl* becomes high all *Group A* buffer contents are loaded onto the buffers belonging to *Group B* sequentially. Consequently, the data stored in *Group B* are loaded onto the output (TxD_{mod}) in a

sequential manner on every edge of the *Clk* signal. For the 64-byte packet case, the output (TxD_{mod}) begins to transmit as soon as TxD has loaded all 64 bytes onto the *Group A* buffers. Therefore, a minimum distance of 64 bytes between the TxD and TxD_{mod} packet commencements is observed, which is the equivalent of 51.2 ns since the XGMII clock speed is 156.25 MHz (6.4 ns), and it takes 8 clock cycles to go through a 64-byte packet. At the same time, it is possible for *Group A* buffers to store a new packet destined for extension header addition. It is important to mention that apart from the buffers of *Group B*, additional storing elements are used at the TxD_{mod} end to hold values for all the extension headers and are connected to the output whenever necessary.

The extension header data are split into two categories. There are constant data that are common in all modified packets (standard fields), and data that change from packet to packet (custom fields) and therefore require a finite amount of processing time. Table 1 lists all header fields for both OWD and OWL measurements, distinguishing between standard and custom. Standard field entries can be permanently stored in memory elements that simply offload their data whenever necessary onto the header addition module output (TxD_{mod}). Custom field entries, however, require the computation of values that will eventually be transmitted as part of the outgoing modified packet.

Table 1: Categorisation between standard and custom field extension header entries

Field	***OWD***	***OWL***
Standard (1 byte each)	Next Header	Next Header
	Header Extension Length	
	Option Type	Header Extension Length
	Option Data Length	
	Pointer	Option Type
	Overflow	
	Flags	Option Data Length
	Reserved	
	DST timestamp (8 bytes)	
Custom	SRC timestamp (8 bytes)	Sequence Number (4 bytes)

SRC timestamp is the system time that has to be transmitted as part of the modified OWD-instrumented packet. Similarly, the Sequence Number is transmitted as part of an OWL-instrumented packet and it is a flow-dependent number that starts from a given value and increments for packets with identical flow characteristics. In fact, the OWL sequence number is fetched from the Flow Table the purpose of which is

explained in the next section. Fortunately, both operations described above take considerably less time than the distance between TxD reception and TxD_{mod} transmission, i.e. 51.2 ns. In other words, the amount of time that elapses from inputting the unmodified packet into the Header Addition module until it begins transmission on TxD_{mod}, provides a length of time that allows for all custom field values to be computed.

The implementation shown at Figure 10 can effectively operate on packets longer than 64 bytes by reusing the existing buffers in an iterative manner, instead of provisioning with additional buffer capacity to accommodate for bigger packet sizes. To some extent, this latter scenario would have been unrealistic since large packet sizes would require an amount of individual buffers that could easily use up all of the FPGA's resources. On the contrary, by employing iterative buffer reuse, as soon as all *Group A* buffers are full they can unload their content onto the buffers of *Group B* and consequently, continue storing same packet data starting from buffer #1 and moving onto higher index buffers. The method is simplified even further by the fact that the extension headers always reside at the same location in a modified packet regardless of the packet's length. The only challenge is in knowing when to connect the *OWL/OWD headers* block of Figure 10 to the TxD_{mod} output. This need only happen once per modified packet, hence if *Group B* buffers are to be used iteratively, the *OWL/OWD headers* block must be ignored after the first run. The total size of each packet can be computed from the value held in the *Payload Length* field (Figure 1(c)) of the main IPv6 header. Consequently, the switch between *Group B* buffers and TxD_{mod} can be instructed to connect the *OWL/OWD headers* block to the output once and then bypass it until the end of the modified packet.

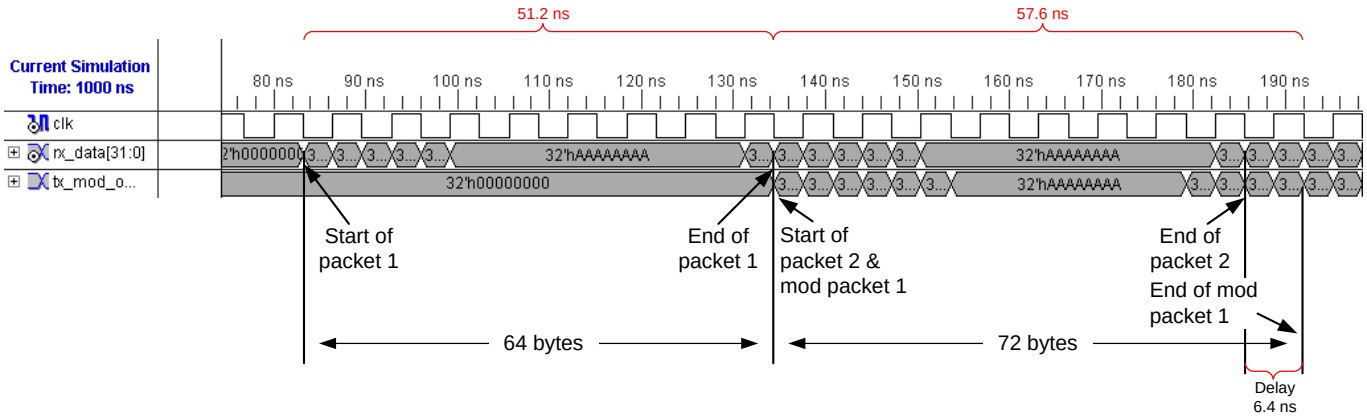

Figure 11: Real-time simulation of OWL header addition for 64-byte packet size

Figure 11 shows a simulation screen of the addition of OWL measurement extension headers on 64-byte packets. *Tx_out* represents the outgoing packet prior to extension header addition whereas *tx_mod* is the same packet modified to include the extension headers. This is the reason why a 64-byte packet prior to header addition will in fact increase in length (72 bytes) post header addition. It can be observed that as soon as the end of *packet 1* is reached, the transmission of the modified packet commences. In case of packets appearing back-to-back on the device, a second packet appears on *tx_out* and will eventually be transmitted with extension headers similar to *packet 1*. Notably, a delay begins to build up between the start of an unmodified packet and a modified one, which is a side-effect that we shall address in later sections.

3.2.4. Flow Table

The purpose of the Flow Table is to hold a number of entries that maintain state information necessary for the OWL measurement. Being a differential measurement, packet loss is computed as the difference in the sequence numbers carried within two successive packets. In this respect, "successiveness" is subject to some flow specification, and although it is most commonly defined in terms of individual microflows, it can also be used in the context of flow aggregates. For example, one can measure the packet loss exhibited by all traffic exchanged between two netblocks, in which case flow classification would have been based on IP source and destination address prefixes only. Each entry is comprised of a series of individual fields, as shown in Figure 12. The first five fields hold packet information identical to what is used to form the traffic filter specification described earlier. Flows are classified based on a combination of source and destination IP(v6) addresses, transport protocol, and transport protocol source and destination ports, which can hold either individual values or wildcards. The next two fields hold the necessary per-flow state information for the implementation of OWL measurement: a sequence number of the last packet seen to belong to the flow specified by this entry, and a timestamp of this packet's arrival time at the instrumentation device. The timestamp is used to identify flow inactivity, and delete the corresponding entry from the flow table, if no packets from that flow have been received for a time interval beyond a certain threshold. Upon reception of a packet that matches the flow specification, the sequence number is incremented by one and this value is inserted within the OWL extension header option.

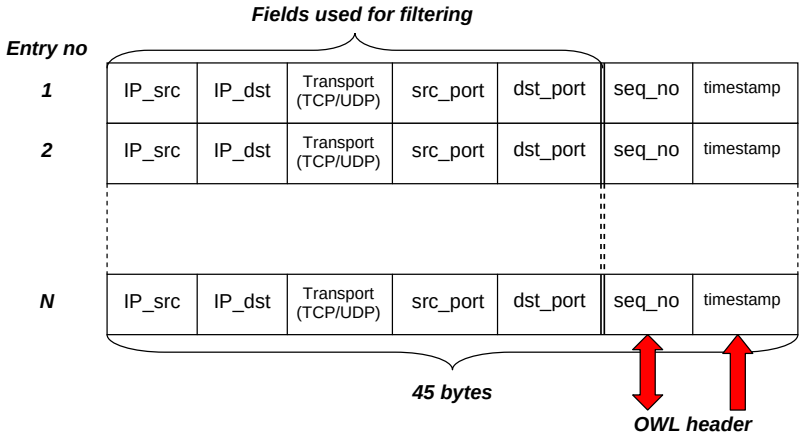

Figure 12: Flow Table entries and fields

The relevant flow table's field is updated accordingly, and so is the timestamp field. If, of course, a flow table entry is not found to match a given packet, a new entry is created and a sequence number is initialised. If a new entry needs to be created when the flow table has reached its capacity, then the flow entry with the largest inactivity interval is being replaced. In case of OWL measurement, flow classification happens in parallel with packet filtering, since the two processes adhere to the same specification. If a packet satisfies the filter specification, then the corresponding flow entry is updated (or created if it does not exist) in the flow table. If, on the other hand, the filtering criteria are not satisfied, the flow table remains unaltered and the corresponding pointer is released. The flow table is periodically updated using an out-of-band process that deletes entries that have been inactive for more than a specified time interval.

Under the current implementation scheme, a maximum of approximately 100 flow table entries could fit in the XC5VLX30T along with the OWL/OWD header addition and extraction mechanisms. For additional storage of flow entries external memory would be required. For instance, taking into account that an individual entry is 45 bytes long, 45 KB would allow for the storage of 1000 additional entries and so on.

3.3. Packet Delay and Loss Header Extraction

Measurement header extraction is implemented at network egress, when an incoming already instrumented packet is received to be forwarded to a neighbouring network. The measurement indicators carried within the packet are amended according to the particular measurement implementation (e.g. for

OWD a second timestamp of packet arrival should be computed, whereas for OWL the TLV's sequence number would only have to be recorded), and are subsequently stored in order to be consumed by higher-level measurement applications, which are outside the scope of this paper.

In terms of complexity and design effort, OWD and OWL extension header *extraction* is a simple task compared to extension header *addition*. Those two operations make up the Extension Header Operations block of Figure 6. Extension header addition comprises the final stage of an outbound packet's journey prior to entering the XGMII interface, whereas extension header extraction is the very first function that an inbound packet will go through just prior to the Reconciliation Sublayer (RS). Figure 13 shows the main functional blocks used for implementing extension header extraction. The inbound packet is propagated onto a Flag Check module and at the same time every 32-bit chunk of data is stored in a different buffer.

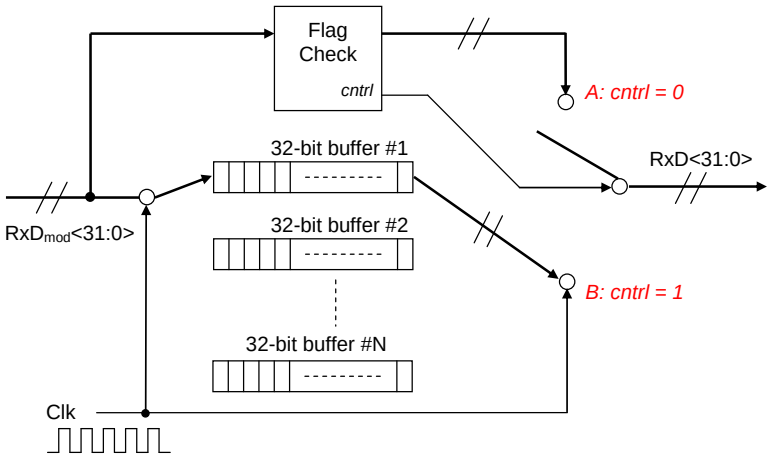

Figure 13: Schematic for header extraction module

The purpose of the Flag Check module is to read the Next Header entry in the inbound packet's IPv6 main header. If this value indicates the presence of a destination options extension header ($=60_{10}$), signal *cntrl* will become logic 1 otherwise it becomes equal to logic 0. As a result, output RxD is connected to either of two switches (*A* and *B*). Switch *A* (*cntrl = 0*) connects the inbound data straight to RxD whereas switch *B* (*cntrl = 1*) connects RxD to a different buffer (starting from buffer #1) on every half cycle. The collection of N buffers is filled by the contents of the packet, including the measurement extension header data. This is then extracted by connecting RxD to all buffers excluding the ones holding the extension

header data. Figure 14 shows the simulated real-time operation of the implementation described so far. *Rx_data2* stands for a 72-byte inbound packet (minimum-sized packet including the OWL header). The presence of the header will activate the removal process, and as a result, *tx_mod* carries the same packet without the extension header. Notice that there is a delay (4 clock cycles) between the two signals since the packet header entry (IPv6 Next Header field) checked by the Flag Check module in order to identify whether an extension header is indeed present is not located at the beginning of the inbound packet. Therefore, packet transmission at the output will begin after this particular entry has been checked and appropriate course of action has been selected.

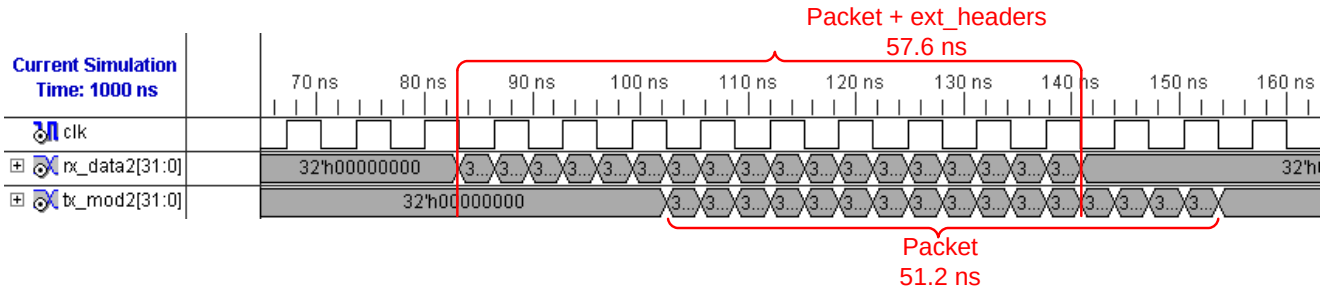

Figure 14: Real-time simulation of OWL header extraction for 64-byte packet size

3.4. Hardware Utilisation and Speed

The particular FPGA device used as a platform for this work is powerful both in terms of speed and hardware resources. First, through some careful designing, it can operate at speeds that surpass the 156.21 MHz threshold imposed by the XGMII interface. Actual synthesis estimate on the maximum clock speed for this design is 183 MHz. This means that the proposed technique can operate with ease between the XGMII interface and the Reconciliation Sublayer (RS). Second, assuming that flow table entry storage will be taken over by a dedicated memory module (such as a 45 KB or more EPROM), the actual hardware resources consumed by the extension header addition and extraction functionality are quite low, as shown in Table 2 which focuses on the FPGA hardware utilisation for OWL measurement on minimum-sized (64-byte) packets. Simulation results for variable packet sizes and OWD measurement show insignificant differences of less than 1% in resource utilisation and are not separately shown here due to space limitations. Before moving onto explaining the information presented in Table 2, it is important to note that a number of different FPGAs were considered in the context of this work, such as the Spartan series and the

Virtex2/Virtex2Pro, however, it was found that the Virtex5 family offers the best results in achieving optimum performance with the least design effort. Spartan FPGAs, in particular, have been developed with economy in mind both in power consumption and cost of purchase. They are, however, considerably resource-limited compared even to one of the most basic Virtex5 FPGAs such as the XC5VLX30. In fact, it has been found that in terms of hardware resources, only a top range Spartan FPGA would be capable of implementing this design, such as the Spartan3E XC3S1600E or Spartan3E XC3S500E. In addition, using the exact same behavioural model of the proposed design (VHDL coded) does not guarantee the same clock speeds for different FPGAs. Spartan and Virtex2 FPGAs achieve poorer results than the Virtex5 FPGA, which suggests a considerably longer design cycle before the implementation meets all performance criteria. Moreover, the selected Virtex5 FPGA is left with considerable amount of hardware resources uncommitted and therefore offers the opportunity to revisit the system in-field and expand its functional capabilities.

Overall FPGA resource utilisation is commonly represented by the *slice logic utilisation* rows. A slice constitutes one of the fundamental FPGA metrics since it directly points to the number of *logic cells* in it. The information presented in Table 2 is split into three different categories in order to show how the utilisation of the device's resources changes according to the implemented functionality. First, it shows the synthesis results obtained through header *addition* measurements which also includes the additional hardware resources required for complementary functions such as filtering. It can be seen that the slice logic utilisation is considerably low compared to the available resources with a 5% occupation of all *slice registers* and 3% usage of all *slice LUTs* (Look-Up Tables). The next column shows the combined slice utilisation of the FPGA resources when both OWL header *addition* and *extraction* are implemented on the device. It is noticeable that the slice logic utilisation remains at significantly low-levels with only 8% of *slice registers* and 4% of *slice LUTs* needed for the implementation of both OWL measurements on the FPGA. Finally, a third entry shows the resource increase when using some of the device hardware for the purpose of implementing the flow table (along with extension header *addition* and *extraction*). The flow table data size is on the order of 450 bytes and what is noteworthy is that although the *slice register* increase is somewhat small (2%), the *slice LUTs* and *slice LUTs used as logic* increase is 7%.

Table 2: Virtex5 XC5VLX30T device utilisation for OWL measurements

	Component	*Utilisation (Addition)*	*Utilisation (Addition, Extraction)*	*Utilisation (Addition, Extraction, Flow table)*
Slice Logic Utilisation	Slice Registers	5%	8%	10%
	Slice LUTs	3%	4%	11%
	Slice LUTs used as logic	3%	4%	11%
	Slice LUTs used a memory	0%	0%	0%
I/O Utilisation	Number of I/Os	116	215	225
	Number of bonded IOBs	31%	58%	61%
Specific Feature Utilisation	Number of BUFG/BUFGCTRLS	6%	6%	6%
	Number of DCMs	25%	25%	25%

Naturally, every doubling of the flow table size would imply a further 7% increase on those two entries, which means that on the FPGA there would be the capability to store approximately 4.5 KB of data. This would leave no more space available to accommodate any possible expansions onto the extension header *addition/extraction* scheme, hence the use of an independent storage unit for that purpose has been considered. If both OWL and OWD measurements were combined on the FPGA, it is estimated that approximately 20% of the *slice registers* and 8% of the *slice LUTs* would be required. This is due to OWL taking up slightly less than half and OWD taking up slightly more than half of those two figures.

It is therefore shown that initial estimates, through FPGA synthesis, support the notion that the method investigated in this paper would be economical to implement on reconfigurable hardware with capabilities equivalent to those of the Virtex5 FPGA device selected for this work. Finally, it is worth mentioning that at the moment many of the FPGA's input/output pins are used as interfacing ports of the top level entity that includes both modules (OWL header *addition* and *extraction*). This has been useful in simulations since it allows for debugging the implementation by offering an inside view as to how various signals in the design behave, but it will not be a necessity in the finalised implementation. One last notable observation with respect to Table 2 is the DCM utilisation. It has been mentioned that the FPGA design presented in this paper, makes use of a single DCM. Only one out of a total of four is currently bounded to the existing design leaving the remaining three available for use in additional expansions on the same chip if so desired.

4. Operational Analysis

In-line measurement instrumentation operates in a similar manner to other widely deployed mechanisms that are applied either end-to-end (e.g. Virtual Private Networks – VPNs, IPSec tunnelling) or in-network (e.g. Multi-Protocol Label Switching – MPLS, IPv4/IPv6 tunnelling) and provide added functionality at the expense of marginally increasing the packet size. All these mechanisms incur an implicit penalty in the maximum bandwidth that can be sustained while additional data is being piggybacked onto data-carrying packets, and they are part of the reason networks are heavily over-provisioned to operate at less than 50% utilisation [14]. The in-line measurement header addition module that operates at network ingress increases the packet size by 8 and 24 bytes, for packet loss and delay instrumentation, respectively. In contrast, however, to the aforementioned relevant mechanisms that require all packets of a given stream to be piggybacked or encapsulated in order to operate, in-line measurement can be equally applicable while operating at a configurable level of granularity to match a network's utilisation. Indeed, the developed filtering and sampling schemes can be exploited to apply measurement instrumentation at a subset of traffic to incur minimal to no impact to the network's operation.

In this section we will examine the worst-case impact of the hardware in-line measurement instrumentation when operating as an integral part of very high-speed routers, based on data collected from simulation analysis. We will also discuss methods to alleviate the impact of the instrumentation process at full line speed without necessarily resorting to tightening the filter specification or decreasing the sampling rate. At the worst case of continuously instrumenting (wildcard filtering and sampling) back-to-back packets arriving at full line speed (10 Gb/s), there is a point in time where outbound packets will have to be dropped. To clearly describe this challenge we shall examine the case of OWL extension header addition on back-to-back, minimum-sized (64-byte) packets. An outbound packet appears at the extension header addition module input, as shown in Figure 15. The time at which a modified packet TxD_{mod} can begin transmission relative to the appearance of its unmodified counterpart TxD at the input of the *Header Addition* block is limited by two factors. First, all filtering stage flags have to indicate that the outbound packet should indeed be modified, as discussed in section 3.2.1. In this process, different parts of the

outbound packet are compared against predetermined values and hence the Header Addition block has to 'wait' until all flags instruct it to begin packet data modification and transmission. Second, the hardware implementation of the Header Addition stage, described in section 3.2.3, specifies that only when all TxD data are loaded onto the *Group A* buffers, signal *cntrl* can allow for their propagation onto *Group B* buffers and consequently onto the output.

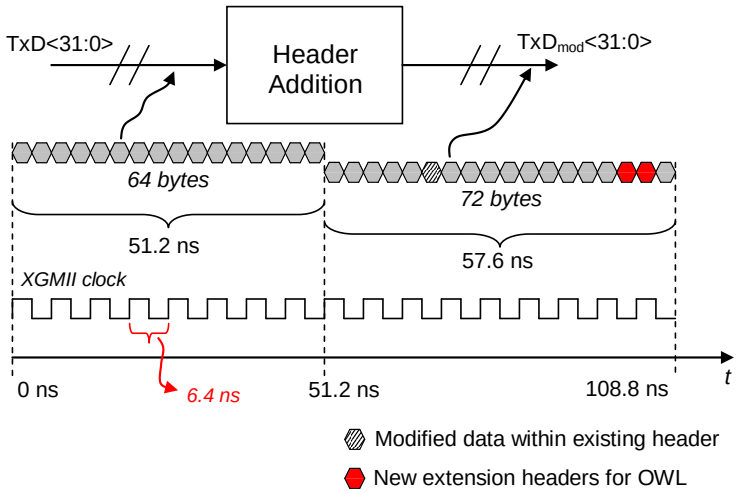

Figure 15: The outbound unmodified and modified packet transmission

Data are transmitted in chunks of 32 bits every 3.2 ns (XGMII interface) and for a 64-byte packet it takes 51.2 ns to complete. Consequently, the modified packet commences transmission after 51.2 and since it is 8 bytes longer than the original packet, it requires 57.6 ns to complete. Figure 16 shows what naturally develops when a sequence of back-to-back packets appear at the Header Addition module input requiring the addition of a measurement extension header. As soon as the reception of *Packet1* ends, the transmission of (the modified) *Mod_packet1* begins. Notably, the start of *Mod_packet1*'s transmission coincides with the start of *Packet2*'s reception. The difference in size, however, between original and modified packets results in a *delay* that increases with time. To better understand the situation we have to revert back to the implementation shown in Figure 10. Every time an unmodified packet completes loading onto the buffer array of *Group A*, signal *Cntrl* toggles and all *Group A* contents are sequentially loaded onto buffer array *Group B*, which also propagates its contents onto the output (TxD_{mod}) in a sequential manner. *Group B* buffers operate in an alternating state mode, first unloading data onto TxD_{mod} and then being updated with new data. However, in case of 64-byte packets, a 51.2 ns cumulative system delay (the length, in units of

time, of the unmodified minimum-sized packet to appear at TxD) disrupts the repetitive alternation between the two states and forces a *Group B* buffer to update its contents twice before unloading data onto TxD_{mod}, which results in a modified packet *(Mod_packet#N)* that contains data from two different packets. This is marked by the occasion where a modified packet *i* begins transmission at the same instant as an unmodified packet *i + 2* is received into the buffers of *Group A*, and will lead to corruption of the modified packet.

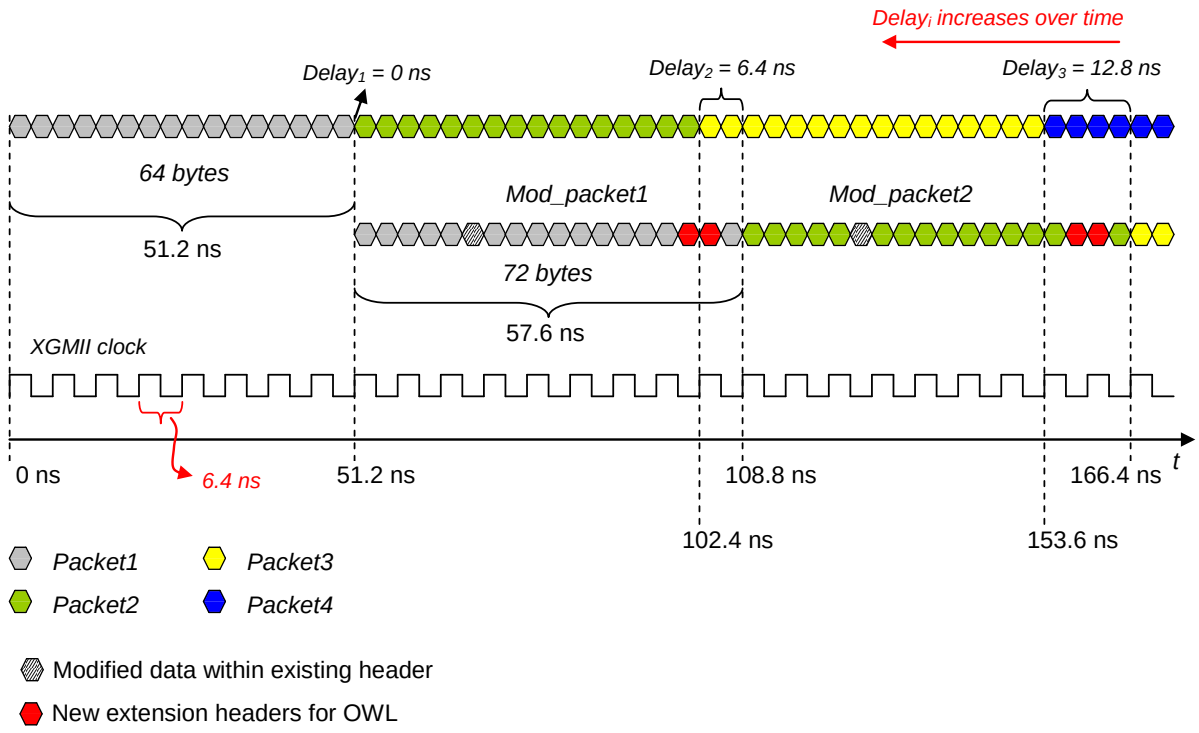

Figure 16: Accumulating delay due to back-to-back packet filtering

Dropping the corrupted packet will reset the system processing delay back to 6.4 ns, continuing accumulation from this value once again. This problem recycles over and over and the frequency at which it will occur is a function of a number of different parameters. For instance, systematic count-based sampling can be used to extend the time before a packet drop occurs. For minimum-sized back-to-back packets at 10Gb/s, instrumentation would result in a packet drop every 409 ns (i.e. every eight back-to-back packets), Naturally, results improve for bigger packet sizes, since the time it takes for packet corruption to occur increases accordingly. For example, instrumentation of back-to-back maximum-sized packets at 10 Gb/s would result in a packet drop every $230 \cdot 10^3$ ns (every 189 back-to-back packets). Figure 17 shows the number of transient packets (both modified and unmodified) until a single packet drop occurs against the delay that develops between outbound packets at the input and output of the Header Addition module,

assuming back-to-back packet arrival at 10 Gb/s, for minimum and maximum-sized Ethernet segments. The figure also shows how many back-to-back packets are required to build up the internal system delay that would result in a single packet drop for different count-based sampling granularities.

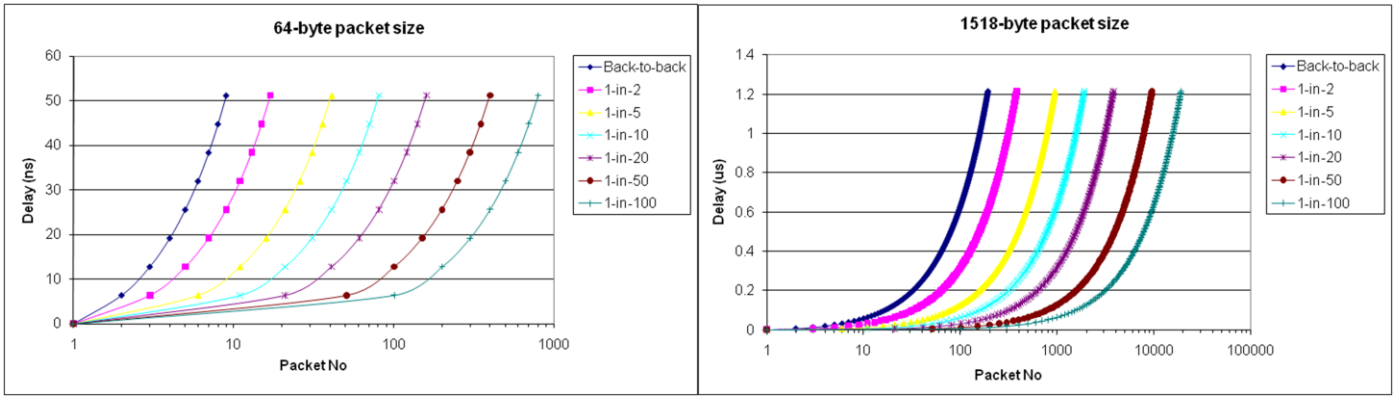

Figure 17: Number of packets through Header Addition module until packet drop occurs

4.1. Maximum Data Rate of Instrumentation

One approach to alleviating packet drops when the system operates at maximum stress is to decrease the rate at which data are provided to the Header Addition module. This can eliminate the problem entirely by virtually expanding the length of unmodified packets arriving for instrumentation at the header addition module, so that their transmission time matches that of a modified packet at 10 Gb/s. Reducing the rate at which 32-bit chunks of packet data are transmitted decreases the network speed. The maximum achievable rates of the hardware implementation for the two in-line measurement modules (OWD and OWL) are summarised in Table 3, for three different packet sizes representative of minimum, medium and maximum-sized Ethernet segments. It is worth noting that the rates below are comparable to the maximum achievable rates within an MPLS network, and much higher than the true data rates achieved by protocol and/or security encapsulation.

Table 3: Maximum allowable data rates for different packet sizes

Packet size (bytes)	*Data rate for OWL*	*Data rate for OWD*
64	8.88 Gb/s	7.27 Gb/s
512	9.8 Gb/s	9.5 Gb/s
1518	9.94 Gb/s	9.84 Gb/s

4.2. Additional storage elements

Additional hardware resources can be used as buffers to store packets as they come into the instrumentation module, and therefore virtually increase the system throughput and lessen the extent to which packet drops occur, when the system operates under extreme stress. As shown in Table 2, the base hardware design uses approximately 10% of resources (Slice LUTs) of the chosen FPGA platform, leaving significant space for extensions that will further reduce the instrumentation impact on throughput. In order to evaluate the benefits of using such a packet storage medium, a distributed RAM has been implemented on the platform FPGA, shown in Figure 18 in conjunction with the overall system. Only fundamental signal lines are highlighted, including write enable (we) and address specifier (addr) signals for the distributed RAM block. This design allows for packets to be directed either to the memory buffer or straight to the Header Addition Module depending on whether they need to be queued or processed directly. When extreme stress conditions arise (discussed in section 4), packets can be stored in RAM rather than dropped. The base design did not use any slice logic for implementing memory, hence the relevant column of Table 2 shows 0% utilisation. However, the FPGA provides 5120 LUTs for memory purposes which in our case can be used for storing excess data prior to instrumentation. Figure 19 shows the size, in bytes, that can be stored in memory over the LUTs used. The maximum number of packet data that can be stored on-chip are 32 KB at the expense of 4096 LUTs. Under the worst case operating scenario of instrumenting back-to-back packets, this would significantly reduce packet drops by three orders of magnitude. As discussed in section 4, OWL instrumentation of minimum-sized (64-byte) back-to-back packets at 10 Gb/s would result in a single packet drop every 409 ns. Using the 32 KB of distributed RAM to store excess data will create a back-pressure mechanism for the header addition module that will extend successful packet instrumentation with no packet drops for $209.5 \cdot 10^3$ ns. Similarly, for maximum-sized packets, this back-pressure mechanism will extend the time for a packet to be dropped from $230 \cdot 10^3$ to $4.9 \cdot 10^6$ ns. As we will discuss in section 5, such burst lengths (on the order of thousands of microseconds) of back-to-back traffic do not occur on today's backbone networks. Therefore, the trade-off between memory hardware resources and storage for packet data can be left at the discretion of particular deployments depending on the application requirements

and traffic demands. At the same time, it should be noted that even if all LUTs for memory are used, a significant part of hardware resources destined for logic utilisation remain available. Analysis has shown that 79% of the slice LUTs used as logic are available for further design additions/modifications or to be used as extra storage elements.

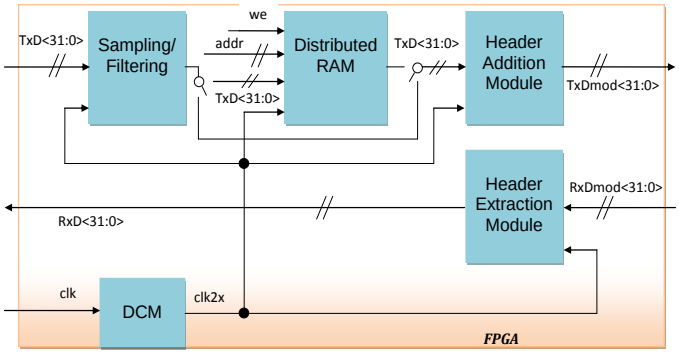

Figure 18: Extended design including RAM module

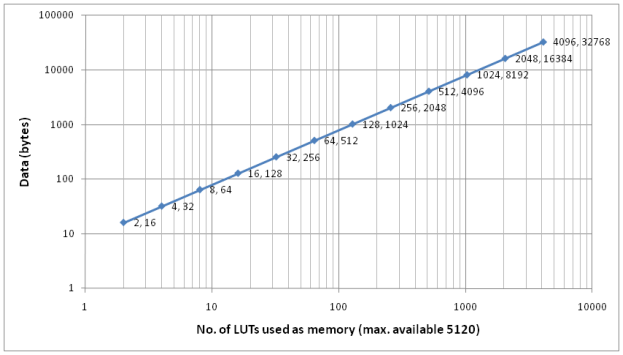

Figure 19: RAM size over implemented LUTs

4.3. Maximum Transfer Unit (MTU) considerations

The addition of measurement extension headers to packets in transit should be done with care not to cause packet fragmentation and negatively impact the performance of instrumented traffic. Fragmentation can be caused if the size of a packet grows bigger than the Maximum Transfer Unit (MTU) of a link along the end-to-end path. In contrast to IPv4 where fragmentation is performed en-route, in IPv6 fragmentation is end-to-end. In order to avoid intermediate routers having to fragment packets, IPv6 end-hosts are required to perform Path MTU (PMTU) discovery prior to establishing a connection, unless they use the guaranteed minimum MTU of 1280 octets [12]. It is reasonable to assume that backbone network topologies support MTUs larger than the guaranteed minimum, which is also evident in Figure 20 that shows maximum-sized Ethernet segments of 1518 bytes being transferred over operational backbone links. When PMTU discovery is employed, then instrumented (network ingress) nodes should advertise a MTU according to equation (1).

$$MTU_{adv} = \min_{i=0...L} MTU_i - H \quad (1)$$

The advertised MTU_{adv} for a network topology consisting of L links should be H bytes less than the minimum MTU of the topology, where H is the size of the measurement header to be added to the instrumented traffic. It is reasonable to assume that within an ISP network, the minimum MTU across all

links is known. Under this assumption, intermediate nodes within a network do not have to be modified, nor do they have to be aware of the instrumentation taking place at the edges, since the MTU advertised by the ingress (instrumented) node guarantees that the addition of measurement headers will not cause packet sizes to exceed the topology's minimum MTU. If at the end of PMTU discovery, MTU_{adv} is the minimum across the end-to-end path then this will be adopted by the end-hosts, which will allow for measurement headers to be inserted to the packets and carried along within the boundaries of the instrumented network topology. If a $MTU < MTUadv$ is adopted by the end-hosts, again edge-to-edge instrumentation within a network can proceed as normal since at network egress the measurement header will be removed. PMTU discovery is one of several techniques that can be used to avoid fragmentation or packet dropping in the presence of mechanisms that increase the packet size in the network such as, for example, MPLS or packet encapsulation/tunnelling. Other, less elegant techniques include Maximum Segment Size (MSS) clamping at network ingress, and routing of baby giant frames (frames slightly longer than the MTU) across the topology. However, detailed description of such mechanisms is outside the scope of this document.

5. Overhead analysis for high-speed backbone network links

In this section we assess the operational impact of the instrumentation process on an actual backbone 10 Gb/s Internet link experiencing a "typical" workload. The typical provisioning rule in effect in most large-scale IP networks is that links are not supposed to exceed 50% utilisation in order to limit queue occupancy and cross-router delay, and also provide robustness to failures. In reality, link utilisation across Tier-1 networks is consistently reported to be far lower than 50% [14], which leaves enough capacity for instrumenting packets at high link speeds using the hardware implementation of in-line measurement presented in this paper. We expect that the virtual data rate reduction incurred by the instrumentation process even at high speed backbone links will be minimal in terms of packets that would have to be dropped or queued. In order to verify this claim, we have used a recent, hour-long high resolution traffic trace provided by CAIDA [2] to analyse the performance of the hardware prototype based on actual packet size and interarrival time patterns. The trace captured all traffic at a 10 Gb/s backbone ISP link connecting Chicago, IL and Seattle, WA [24], and it includes over 716 million records of individual IP packet arrivals,

each consisting of a nanosecond resolution timestamp and the IP packet size S_p in bytes. We have processed the trace to calculate packet interarrival times and to adjust S_p to include a full Ethernet segment, by choosing $S_p=max(64, S_p+18)$. The minimum packet interarrival time *IA* between two (back-to-back) packets *p-1* and *p* over a 10 GigE link is linearly analogous to the full Ethernet segment size of *p* at 10 Gb/s, $\min(IA_{p-1,p})=0.8*S_p$ (ns). Hence, assuming Ethernet segment sizes from 64 to 1518 bytes, the theoretical minimum packet interarrival times for 10 Gb/s vary from 51.2 to 1,214.4 nanoseconds. The hardware in-line measurement process at 10 Gb/s adds a constant per-packet transmission time *C* of 6.4 and 19.2 ns for OWL and OWD instrumentation, respectively. Therefore, for two successive packets being instrumented, a processing delay component D_p is added if the second packet arrives at a rate less than the minimum transmission time plus the constant component *C*, as shown in equation (2) below.

$$\begin{cases} D_p = 0.8*S_p + C - (t_p - t_{p-1}), & t_p - t_{p-1} < 0.8*S_p + C \\ D_p = 0, & t_p - t_{p-1} \geq 0.8*S_p + C \end{cases} \tag{2}$$

Assuming a series of *k-i+1* packets arriving within a time interval $[t_{i-1}, t_k]$, all of which are instrumented with in-line measurement indicators (wildcard filter specification), the processing delay component D_p (in ns) will add-up depending on the arrival rate of the successive packets according to equation (3).

$$\sum_{p=i}^{k} D_p = 0.8* \sum_{p=i-1}^{k-1} S_p + (k-p)*C - (t_k - t_i) \tag{3}$$

Of course, a negative cumulative delay component simply means that packets arrived at sufficiently large spacing to accommodate for the insertion of the in-line measurement headers and their subsequent transmission at 10 Gb/s, and can be therefore set to zero.

We have computed the processing delay component that would accumulate in the system if wildcard traffic instrumentation had taken place at a typical backbone 10 Gb/s Internet link. Even though it can be assumed that links are over-provisioned, the burstiness of traffic is important in determining the system's performance and in quantifying the levels of resulting packet drops, if applicable. We first examine the presence and intensity of packet trains in the trace. A packet train is a burst of packets arriving back-to-back

with minimum interarrival time between them determined by the capacity of the medium and the packet size. Overall, packet trains comprise 3.7% of the entire trace, with the rest of the packets arriving with longer spacing. Figure 20 shows the Cumulative Distribution Functions (CDF)s of the length of such bursts and their duration, and also the packet (Ethernet segment) size distribution for the observed trace.

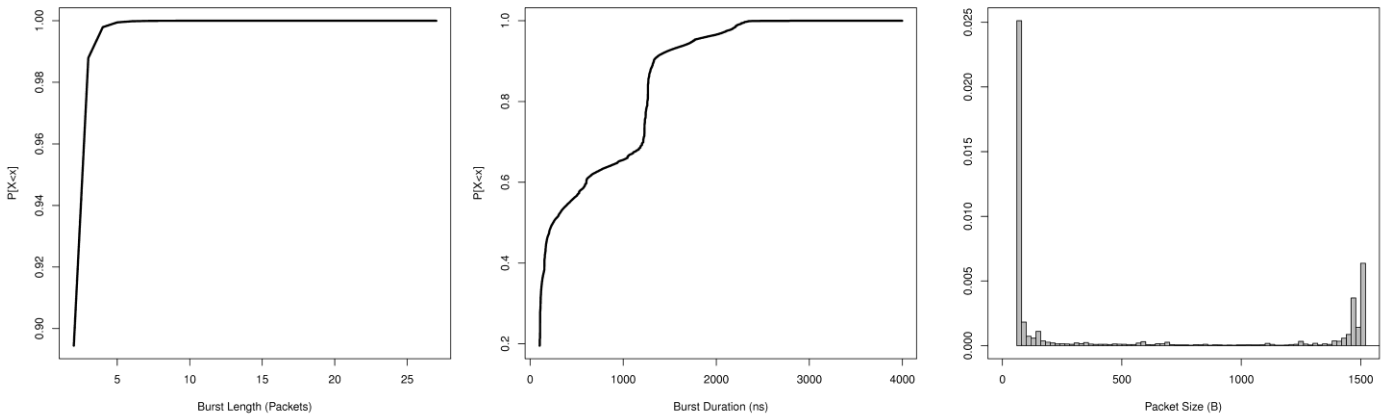

Figure 20: Packet Trains and Packet Size Distributions for the CAIDA dataset; 15/05/2008; 12:59:07 – 13:58:45

It is apparent that long packet trains appear extremely rarely. The vast majority of packet trains are of size two (89.4%), meaning that only two packets arrive back-to-back at a given instant. 99% of the burst lengths is less than four packets, and there is only one occasion in the entire trace where 26 packets arrived back-to-back. The duration of each burst is therefore mainly bound by the size of two successive segments, which are very rarely of maximum size. 95% of burst duration is less than the time required to transmit/receive two maximum-sized Ethernet segments back-to-back (2,428.8 ns). The packet size distribution exhibits the typical bimodality at minimum and maximum packet sizes with far fewer occurrences of medium-sized datagrams, reinforcing the argument that although the majority of packets are minimum-sized (e.g. TCP acknowledgments), the majority of bytes (~73% in our case) are transferred within maximum-sized packets [3]. These two latter observations (the extremely rare occurrence of back-to-back maximum-sized packets which carry the majority of bytes) are very important for in-line traffic instrumentation, since they imply that the main data-carrying packets can be instrumented completely un-intrusively. Overall, the low burstiness levels imply that the hardware-based in-line measurement modules at the edges of an ISP network can hardly impact its capacity and forwarding operation. We will now quantify this argument through detailed results regarding the impact of the accumulated system delay

caused by the instrumentation process. Figure 21 shows the Complementary CDFs (P[X>x]) of the system's accumulated delay when a packet arrives at the OWL and OWD hardware instrumentation modules, respectively. Each plot shows the overall distribution of cumulative delay for all packet arrivals (irrespective of the arriving packet's size), for minimum-sized, and for maximum-sized packet arrivals.

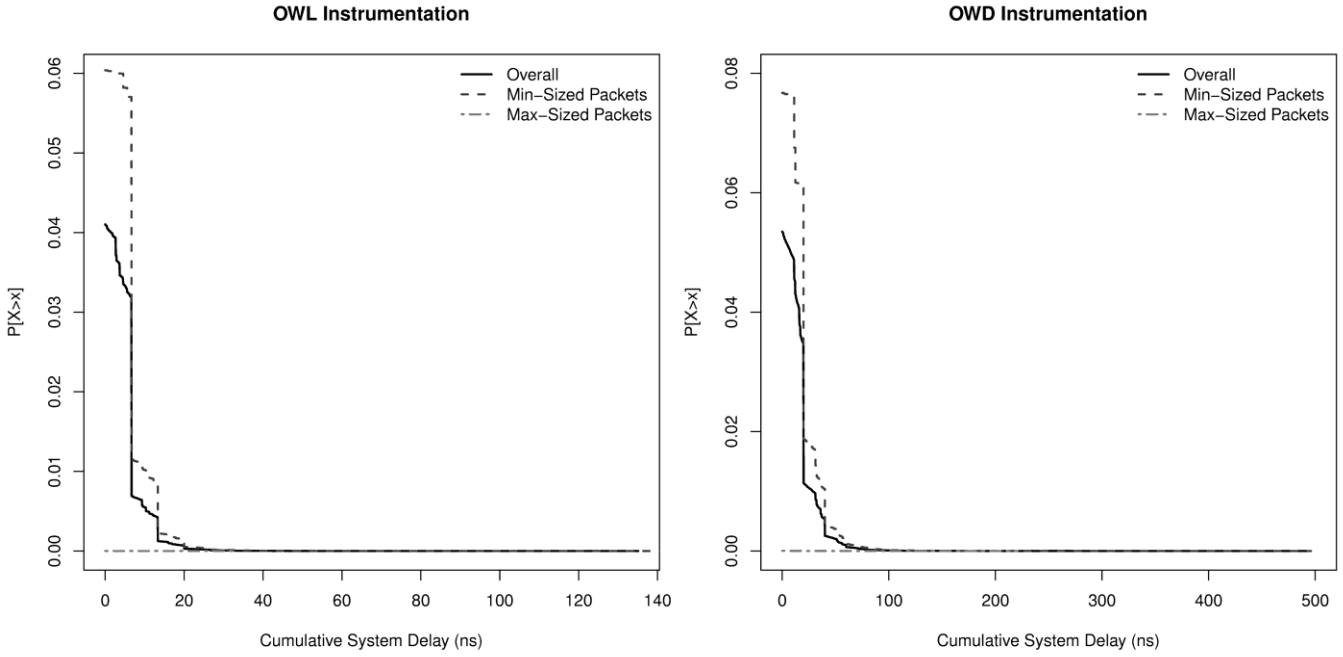

Figure 21: Cumulative Processing Delay for OWL and OWD in-line measurement hardware instrumentation

It is clear that the system doesn't accumulate any internal delay for most packet arrivals. The interarrival time of 95% and 94% of the packets is sufficiently long for the system to absorb any delay caused by the instrumentation of the previous packets with OWL and OWD measurement headers, respectively. The mean cumulative delay is 0.29 ns for OWL and 1.15 ns for OWD instrumentation, well below the values that would cause a packet to be dropped due to corruption (see section 4). There is however a tail that reaches larger values for both types of instrumentations worth investigating. The cumulative system delays that would cause a subsequent packet to be dropped or queued are between 51.2 ns (when the next packet arriving is a minimum-sized –64-byte– segment) and $230 \cdot 10^3$ ns (when the next packet arriving is a maximum-sized –1518-byte– segment). According to the two distributions, the percentage of packets that would need queuing or dropping due to the system's accumulated delay is 0.0001% and 0.046% for OWL and OWD instrumentation, respectively, an admittedly very low fraction of traffic and assuming that every

single packet is instrumented. It is evident that these values can be reduced to virtually zero, if the developed sampling mechanism is even very moderately exploited. Since different values of accumulated delay can cause a packet to be dropped depending on its size, it helps examining the delay distributions with respect to different packet sizes, in order to estimate how many packets would need to be dropped or queued. The two dashed curves in each plot of Figure 21 show the distribution of accumulated system delay prior to the arrival –and subsequent instrumentation– of minimum (64-byte) and maximum-sized (>1400 bytes) packets. The reason we have chosen these two packet sizes/range is the bimodality of the packet size distribution (Figure 20). For maximum-sized packets, there would be zero packet drops for either type of in-line measurement instrumentation, verifying the claim made earlier that the main data-carrying packets arrive at sufficient spacing to accommodate for OWL and OWD instrumentation of every packet. Packet trains, when they appear, seem to primarily consist of minimum-sized packets, and hence the two relevant curves of cumulative system delay show a resemblance. Indeed, the percentage of minimum-sized packets that would need to be dropped or queued if wildcard traffic instrumentation was applied, exactly matches the overall percentage at the given accuracy level. Over 95% of packet drop instances would consist of a single packet for either type of instrumentation. It is evident that even a fraction of the developed on-chip memory for buffering purposes (described in section 4.2) easily absorbs all packet drops for this typical workload. Moreover, if all 32 KB were used for storing excess load arriving back-to-back, the burstiness levels would have to increase by three orders of magnitude before a single packet is dropped, and assuming that all packets are instrumented. This is highly unrealistic given the provisioning practices in today's backbone networks. At the same time, in-line measurement offers the opportunity of not having to instrument every single packet over a link. Indeed, instrumenting every minimum-sized packet would be an unnecessary overkill since it is the main data-carrying (larger) packets that are of primary interest when assessing traffic performance. Even so, it is evident that overall, and given a realistic backbone 10 Gb/s link's traffic load, the hardware in-line measurement implementation is a very low overhead and cost-effective instrumentation mechanism.

6. Conclusions

The multi-service and pervasive nature of next generation networks that will interconnect myriad devices urges for ubiquitous measurement systems that will operate natively at an always-on manner, and will truly assess traffic performance in short time scales and at multiple levels of granularity. The specification of IPv6 adopts an extensible design through the selective processing of added functionality that can be exploited to enable measurement instrumentation of traffic as a native part of the network protocol stack.

In this paper we have presented a hardware implementation of in-line measurement, a multi-point mechanism that instruments data-carrying packets and conducts direct per-packet performance measurement. A FPGA-based system has been built to perform traffic instrumentation at 10 Gb/s, and to therefore enable in-line measurement to be effectively deployed even across Tier 1 very high-speed backbone networks. The in-field re-programmability, the high clock speeds, and the significant amount of hardware resources found in today's FPGAs constitutes them ideal platforms for such developments, since they can be calibrated for optimum performance in-field, operate at very high speeds, and offer space for future improvement and expansion of existing designs.

Our implementation demonstrated that in-line measurement operations can be easily accommodated by moderate FPGA architectures and can be integrated in 10 Gb/s hardware controllers. Using a Virtex5 FPGA, we have achieved very low slice logic utilisation values leaving a considerable amount of hardware resources uncommitted for potential expansion of the system's functional capabilities. All the different combinations of measurement (one-way delay and loss) and instrumentation point (addition and extraction of measurement information) functionality can be seamlessly integrated into a single design. We have analysed the main operational characteristics of our implementation and have quantified the worst-case overhead that can be incurred by the inherent attributes of in-line measurement, namely the marginal size increase for instrumented datagrams. Based on rigorous analysis of recent traces from a 10 Gb/s operational backbone link, we have demonstrated that the deployment of such hardware instrumentation system would have an absolutely minimal impact on the operation of a typical high-speed backbone network workload.

References

[1]. J. Aweya, IP Router Architectures: An Overview, International Journal of Communication Systems, Vol. 14(5), pp 447-475, John Wiley & Sons, March 2001

[2]. CAIDA: The Cooperative Association for Internet Data Analysis, on-line resource, available at: http://www.caida.org/home/

[3]. K.C. Claffy, G. Miller, K. Thompson, The Nature Of The Beast: Recent Traffic Measurements From An Internet Backbone in Proceedings of the eighth Annual Conference of the Internet Society (INET'98), Geneva, Switzerland, July 21-24 1998

[4]. D.D. Clark, C. Partridge, J.C. Ramming, J.T. Wroclawski, A Knowledge Plane for the Internet, ACM SIGCOMM'03, Karlsruhe, Germany, August 25-29, 2003

[5]. D.D. Clark, The Design Philosophy of the Darpa Internet Protocols, , ACM SIGCOMM '88, Stanford, California, USA, August 16-19, 1988

[6]. S. Deering, R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, IETF Network Working Group, Request For Comments (RFC) 2460, December 1998

[7]. C. Estan, K. Keys, D. Moore, G. Varghese, Building a Better NetFlow, ACM SIGCOMM'04, Portland, Oregon, USA, August 2004

[8]. C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, F. Tobagi, Design and Deployment of a Passive Monitoring Infrastructure, Passive and Active Measurement Workshop (PAM2001), Amsterdam, NL, April 23-24, 2001

[9]. G. Huston, IPv4 Address Report, on-line resource: http://www.potaroo.net/tools/ipv4/index.html

[10]. IEEE Ethernet Working Group, on-line resource: http://www.ieee802.org/3/

[11]. W. Matthews, L. Cottrell, The PingER Project: Active Internet Performance Monitoring for the HENP Community, IEEE Communications Magazine, May 2000

[12]. J. McCann, S. Deering, J. Mogul, Path MTU Discovery for IPv6, IETF Network Working Group, Request For Comments (RFC) 1981, August 1996

[13]. N. McKeown: A Fast Switched Backplane for a Gigabit Switched Router, Business Communications Review, vol. 27, no. 12, December 1997

[14]. A. Nucci, K. Papagiannaki, Design, Measurement and Management of Large-Scale IP Networks, Cambridge University Press, ISBN: 978-0521880695, December 2008

[15]. NetFPGA, on-line resource: http://www.netfpga.org/

[16]. D. P. Pezaros, D. Hutchison, F. J. Garcia, R. D. Gardner, J. S. Sventek, In-line Service Measurements: An IPv6-based Framework for Traffic Evaluation and Network Operations, in Proc. NOMS'04, Seoul, Korea, April 19-23, 2004

[17]. D.P. Pezaros, M. Hoerdt, D. Hutchison, Low-Overhead, Native End-to-end Performance Measurement for Next Generation Networks, under submission

[18]. D.P. Pezaros, M. Sifalakis, D. Hutchison, On the Long-Range Dependent Behaviour of Unidirectional Packet Delay of Wireless Traffic, in Proceedings of IEEE Global Telecommunications Conference (IEEE GLOBECOM'07), Performance Modelling, QoS and Reliability Symposium, Washington, DC, November 26-30, 2007

[19]. F. Schneider, J. Wallerich, A. Feldmann, Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware, in Proc. Passive and Active Measurement Conference (PAM2007), LNCS 4427, pp. 207-217, Louvain-la-neuve, Belgium, April 5-6, 2007

[20]. J. Shafer, S. Rixner: A Reconfigurable and Programmable Gigabit Ethernet Network Interface Card, Technical Report, Department of Electrical and Computer Engineering, Rice University, December 2006

[21]. W. Stallings, SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, Addison-Wesley Longman, Reading, MA, 1998

[22]. W.N. Venables, B.D. Ripley, Modern Applied Statistics with S, Fourth Edition, Springer Science+Business Media, ISBN 9780387954578, 2002

[23]. Virtex-5 Family Overview: LX , LXT and SXT platforms, technical report, version 3.2, September 2007

[24]. C. Walsworth, E. Aben, K.C. Claffy, D. Andersen, The CAIDA Anonymized 2008 Internet Traces – 15th May 2008, on-line resource: http://www.caida.org/data/passive/passive_2008_dataset.xml