Low-Overhead End-to-end Performance Measurement for Next Generation Networks

Dimitrios P. Pezaros, *Member, IEEE*, Mickaël Hoerdt, and David Hutchison, *Member, IEEE*

***Abstract*— Internet performance measurement is commonly perceived as a high-cost control-plane activity and until now it has tended to be implemented on top of the network's forwarding operation. Consequently, measurement mechanisms have often had to trade relevance and accuracy over non-intrusiveness and cost effectiveness.**

In this paper, we present the software implementation of an in-line measurement mechanism that uses native structures of the Internet Protocol version 6 (IPv6) stack to piggyback measurement information on data-carrying traffic as this is routed between two points in the network. We carefully examine the overhead associated with both the measurement process and the measurement data, and we demonstrate that direct two-point measurement has minimal impact on throughput and on system processing load. The results of this paper show that adequately engineered measurement mechanisms that exploit selective processing do not compromise the network's forwarding efficiency, and can be deployed in an always-on manner to reveal the true performance of network traffic over small timescales.

***Index Terms*—Computer Networks, Computer Performance, Next Generation Networking, Computer Instrumentation, Network Measurement**

I. Introduction

The increased diversity of Next Generation Networks (NGN)s in terms of interconnection technologies, devices and services necessitates the evolution of automated self-* properties for problem diagnosis and operational optimisation [1]. This is in contrast to the original design philosophy of the Internet protocols that placed accountability of resource usage towards the end of their priority list [2]. The fundamental prerequisite for automated network management is the existence of ubiquitous and always-on mechanisms to accurately measure the temporal performance of traffic routed over the infrastructure, in short timescales. Such mechanisms will need to embrace the implementation of an extensible set of performance metrics, in order to capture the diverse Quality of Service (QoS) requirements of the numerous traffic types, and their consequential dependence on different factors of performance degradation. Although Internet measurement research has received much attention during the last decade, a disproportionally small amount of work has been devoted to the design of mechanisms that will directly measure the data-carrying traffic performance. *Active* measurement systems that directly probe a path to assess its temporal performance characteristics are based either on synthetic traffic that does not necessarily reflect the performance of the actual data-carrying traffic [3], or on transport/application-layer mechanisms that render them applicable only to a subset of the traffic [4]. At the same time, *passive* monitoring infrastructures that observe the operational traffic at a single point in the network need to correlate and post-process immense amounts of data before conducting a conclusive performance measurement, and they therefore operate at long timescales [5][6][7]. Active and passive measurement mechanisms cannot be easily extended to operate at an Internet-wide scale and in an always-on manner, mainly due to the overhead they incur either on the network (e.g. additional probe traffic) or on system resources (e.g. tracing traffic at gigabit speeds requires dedicated equipment [8]). Clearly, the ever expanding Internet is in need of more pervasive mechanisms that will be able to ubiquitously measure the performance of the data-carrying traffic itself, in an extensible and configurable manner.

In this paper, we argue that such measurement instrumentation will need to be a native part of the network stack and to operate *in-line*, embedding measurement indicators into the existing operational traffic and hence directly assessing its performance Although traditional active and passive measurement systems do not touch the data-carrying traffic in order to minimise overhead, we argue that the overhead they incur results from their inherent limitation of not being a native part of the network's main forwarding mechanism. We present an in-kernel software prototype of *in-line measurement*, the first mechanism that uses native network-layer structures of the next generation Internet Protocol (IPv6) to encode measurement information into the data-carrying traffic [9]. We show how minimal measurement modules can be completely integrated with the network stack of end-systems, and we demonstrate that properly-engineered measurement mechanisms can be efficient, have minimal impact on traffic performance, and at the same time be part of an extensible framework that can implement numerous

Manuscript received December 16, 2009; revised June 03, 2010; approved by IEEE Transactions on Network and Service Management Editor R. Stadler.

D. P. Pezaros is with the School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, UK (phone: +44-141-330-6051; fax: +44-141-330-4913; e-mail: dimitrios.pezaros@glasgow.ac.uk).

Mickaël Hoerdt was with the Computing Department, Lancaster University, Lancaster, LA1 4WA, UK (e-mail: mickael.hoerdt@gmail.com).

David Hutchison is with the Computing Department, Lancaster University, Lancaster, LA1 4WA, UK (e-mail: dh@comp.lancs.ac.uk).

performance metrics. In-line measurement adopts two fundamental principles that make it particularly applicable: *modularity*, to enable the separation of the traffic instrumentation (real-time) process from the rest of the measurement functionality, and *selective* (measurement) *processing* at pre-identified network nodes as opposed to hop-by-hop. We have particularly focused on the rigorous performance evaluation and the exact quantification of the system cost involved in performing the real-time traffic instrumentation. In contrast to software router performance, which has been extensively studied [10][11], we concentrate on the end-to-end mode of operation where traffic is sent and received from user-space applications, and show that the impact of a software-only in-line measurement prototype on end-to-end traffic throughput is small and statistically insignificant. Throughput is mainly bound by the minimum-sized packets and by the end-system user/kernel context switching. We demonstrate that in-line measurement incurs an overhead which is close to two orders of magnitude less than that incurred by traditional active and passive approaches. In addition, we show that the overall processing impact of the instrumentation process is also statistically insignificant, and we quantify the exact processing time spent on each measurement sub-routine. We conclude that in-line end-to-end network measurement that can reveal the actual traffic-perceived performance over short time-scales is a flexible and low-overhead mechanism, and can operate in an always-on manner as part of the end-systems' protocol stacks.

The remainder of this paper is structured as follows: Section II describes the design of in-line measurement and its different modes of operation, and section III describes the in-kernel implementation of a set of measurement modules. Section IV compares the measurement data overhead of in-line measurement with active and passive measurement systems, and Section V quantifies the impact of the measurement process to end-to-end application throughput. Section VI analyses the processing overhead incurred on the instrumented end-systems. Section VII discusses the benefits of in-line measurement and its implications on network operations and management. Section VIII outlines related work, and section IX concludes the paper.

II. IN-LINE MEASUREMENT

The traditional separation between data and control planes in the Internet has not allowed the integration of measurement mechanisms with the network' main forwarding operation. The protocol stack is strictly standardised and hence any attempt to introduce new structures below the application layer to carry measurement information along the data-carrying traffic would require a major rework of virtually all networked systems. Options that could potentially be exploited exist in a number of protocol layers; however, they have not been originally designed for measurement and are severely restrictive. For example, TCP provides a timestamp option which is, however, not adequate for high precision timing data required for packet delay measurements. Defining a new TCP option to carry measurement indicators is an alternative yet not an ideal one due to the limited space and also its applicability only to a subset of traffic (TCP). The convergent network layer that could potentially be exploited for ubiquitous traffic instrumentation does not allow for arbitrary optional structures to be piggybacked on IP datagrams. Only a set of standardised options exists, and moreover these need to be processed en-route by every IP node. Therefore, not only can new options not realistically be implemented at an Internet-wide scale, even if they did, option-carrying traffic would most certainly exhibit different en-route processing from the rest of the traffic. The introduction of a 'thin layer' – similar to the one suggested in [12] – between the existing network and transport layers could provide for the necessary structures and encoding to carry measurement indicators within the data-carrying traffic, but it would not maintain backward compatibility, since it would require changes to the IP stack of –at least – all end-devices.

The Next Generation Internet Protocol however, overcomes these limitations by introducing a 'thin layer' natively within the ubiquitous network layer. IPv6 adopts the notion of selective network-layer processing through the extension headers concept, and in particular the *destination options* extension header. The protocol specifies general formatting and alignment requirements and leaves options to be defined by programmers and engineers. Destination options are inserted after the main IPv6 header at a source node and are processed only by the packet's ultimate or explicitly pre-identified destination nodes. This is achieved by encoding IPv6 extension headers as an intermediate layer between network and transport, identified through a unique *next header* (protocol) number. Hence, the presence of a destination options extension header within a datagram will not impact its processing by intermediate nodes which will only process the main IPv6 header and will not examine the next (extension) header.

In-line measurement [9] is a novel point-to-point mechanism that exploits these native IPv6-layer structures to instrument data-carrying traffic with measurement options, and to integrate measurement functionality with the network's main forwarding mechanism. It is realistically applicable over the IPv6 Internet in an always-on manner, due to its modularity and selective measurement processing. Modularity ensures minimal operational overhead and independence from particular measurement infrastructures and processes. Only the minimal traffic instrumentation process is implemented as part of a system's network stack, while higher-level applications can be developed independently. In addition, distinct measurement modules implement different performance metrics, again ensuring minimal additional processing. Selective processing, which is inherited from the design of IPv6, ensures minimal impact on instrumented traffic and identical treatment with the rest of the traffic by the forwarding IPv6 nodes.

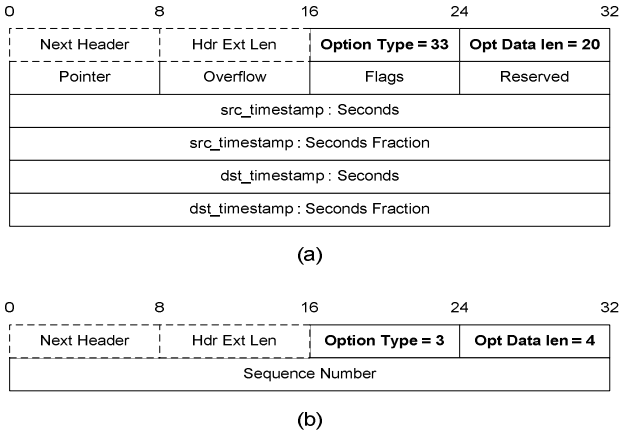

Fig. 1. Unidirectional (a) packet delay and (b) packet loss IPv6 destination options

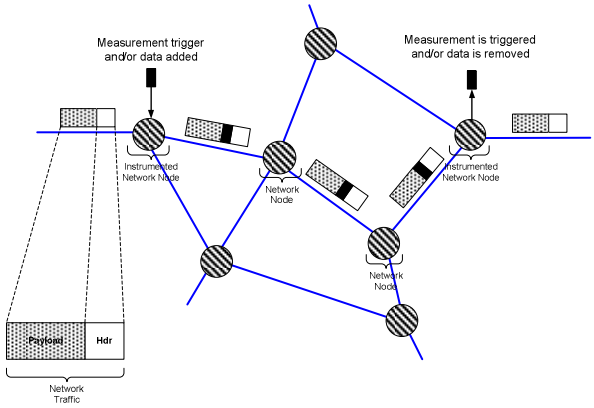

Fig. 2. In-line Measurement Operation

The alleviation of per-hop (measurement) option processing ensures that additional operations take place only at specific pre-identified systems which can be adequately provisioned and not impact traffic performance. We have defined two IPv6 destination options to measure end-to-end unidirectional packet delay, and packet loss, respectively. Their fields, byte alignment and their encapsulation in an IPv6 destination options extension header are shown in Fig. 1. Each IPv6 option is identified by a unique *option type* byte whose encoding specifies the action to be taken by a node if it is supposed to process an option that it does not support. Therefore, not only selective processing but also backward compatibility is preserved. This is an important aspect since experimental options can be deployed before (or while) being standardised. The two options shown in Fig. 1 are representative of the different modes of operation assumed by the in-line measurement process, in a number of ways.

Unidirectional delay is a stateless per-packet measurement which involves two nodes independently recording system time. Similar to timestamp representations in UNIX and NTP, two (unsigned) 64-bit timestamps are encoded within the option to record time at the source and the destination of an instrumented path, respectively. The 32-bit seconds field spans about 136 years, while the 32-bit fraction field allows for a maximum time resolution of around 232 picoseconds, well below the resolution of today's end-systems (microseconds to nanoseconds) [13]. In order to maintain compatibility with NTP, we have kept the same prime epoch.

On the other hand, packet loss is a stateful measurement, since packets are tagged with a network-level sequence number before departure from a source node with respect to some flow specification. Delay measurement consists of two independent measurement actions taken at the source and destination nodes, and hence the option's content is amended at both nodes, whereas for packet loss the destination node simply records the option without amending its content. Finally, the packet loss measurement option demonstrates the smallest possible structure that can be encoded as an IPv6 option due to the protocol's alignment requirements [14].

Fig. 2 shows the operation of in-line measurement on data-carrying traffic. At the source of an instrumented path, a measurement option-bearing header can be constructed and inserted into the packet. Upon arrival at the destination, the presence of the header will trigger a direct measurement action, implementing the relevant performance metric. The self-contained header can then be extracted from the packet and consumed by higher-level measurement applications. It is important to note that intermediate network nodes treat instrumented traffic identically with the rest of the traffic, with no need to be aware of the measurement process. Also, in-line measurement instrumentation is equally applicable end-to-end and edge-to-edge. Packets can be transparently instrumented between ingress and egress nodes of network topologies, and point-to-point metrics such as packet delay and traffic matrices can be directly measured instead of being approximated or computed offline. However, such in-network instrumentation would require hardware support in order to accommodate the high network speeds and be integrated with routers' data-path operation. We have presented a hardware-assisted in-network implementation of in-line measurement in [15].

III. IN-KERNEL MEASUREMENT MODULE IMPLEMENTATION

A prototype in-line measurement system that demonstrates the mechanism's operation as an integral part of the protocol stack has been implemented for Linux 2.6 kernels. The core instrumentation functionality has been implemented as a set of Loadable Kernel Modules (LKMs) that can be linked to a running kernel on-demand. This design provides the efficiency of deploying measurement functionality as part of the OS kernel, and at the same time employs modularity to minimise the actively used processing logic and its operational overhead, by loading only the necessary components to perform a certain type of measurement. Basic instrumentation is unidirectional, and hence an end-system can be the source of an instrumented path, the destination, or both, depending on which measurement modules are loaded at a given time.

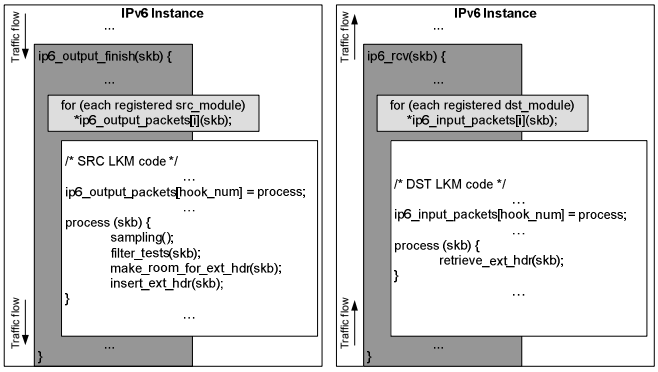

Fig. 3. Generic operation of a SRC and a DST measurement module

Irrespective of the particular measurement implemented, there are distinct operations performed at the source and the destination of an instrumented path by the corresponding measurement modules. The *source* is responsible for initiating the measurement by creating the appropriate header and piggybacking it to outgoing packets. The presence of such header within an incoming packet triggers a direct end-to-end measurement at the *destination*, where – depending on the measurement – the header is amended or simply recorded and then extracted from the packet to maintain full opaqueness from the higher layers. Fig. 3 shows the generic operation and functional decomposition of the SRC and DST measurement modules. The only static modification to the Linux kernel has been the insertion of two hooks to operate on incoming and outgoing packets from within the IPv6 instance's entry and exit functions, respectively. The hooks pass the socket buffer that represents and manages the packet in question to the SRC and DST modules installed at any given time. The provision of measurement instrumentation at the border of the kernel's IPv6 instance offers a number of advantages. First, it allows timestamps to be inserted before a packet spends considerable processing time in the protocol stack of the systems' kernel (although buffering at the device level as well as interrupt coalescing can still distort true network time). And second, it allows for the transparent instrumentation of packets regardless of their ultimate source and destination systems. Although this paper focuses on the end-to-end inline measurement instrumentation, the functions can be equally used to instrument datagrams as they are forwarded at an intermediate system operating as a software router.

A generic SRC measurement module implements a number of functions that enable sampling, filtering, packet handling, and insertion of the measurement indicators to outgoing packets. Two systematic sampling schemes have been deployed to instrument *one-in-N* packets and at *most one packet every M microseconds*, respectively. A five-entry filter specification enables selective traffic instrumentation based on IPv6 source and destination addresses, transport protocol, and transport layer source and destination ports. Sampling and filtering specifications can be altered dynamically by a user process through a system call to the kernel module while the latter is loaded. A shared control structure is used to (re)set the filtering and sampling specifications which will subsequently be read by the module code before attempting to instrument the next datagram. When a packet satisfies the given filtering and sampling criteria, the necessary space is created to the managing socket buffer to accommodate the size of the IPv6 extension header related to the corresponding measurement. If necessary, the buffer's headroom is grown by the appropriate number of bytes (*skb_cow*) and the data area is expanded (*skb_push*) towards the head of the socket buffer accordingly [16]. Finally, the corresponding in-line measurement header is created, updated with the relevant values, and inserted between the main IPv6 and the upper layer header of the packet. For the unidirectional delay measurement, a system call is used to read the clock counter structure and return a 64-bit microsecond timestamp from the start of the UNIX era, which is then converted to NTP format (a delta between the epoch times is added to the seconds field, and microseconds are converted to second fraction) and inserted in the *src timestamp* fields of the corresponding measurement header. The *dst timestamp* fields are initialised to zero. It is worth noting that for a two-point measurement implementation, the *dst timestamp* field need not be initialised at the source and carried along with the packet. It can instead be added at the destination. However, this design choice was made in accordance to the IPv6 specification which states that the source constructs the entire extension header [14]. At the same time, this design maintains opaqueness for a potential edge-to-edge instantiation where the header is amended at an intermediate node and it is then carried within the packet up to its ultimate destination.

For the packet loss measurement, the SRC module maintains flow state using a linked structure whose unique elements are identified by a five-tuple identical to the one used for the filter specification. It is worth noting that each entry of the flow table can be set to an individual value or to wildcard, and therefore entries can resemble individual transport protocol source-destination pairs (microflows) or flow aggregates (e.g. all traffic routed to a particular destination, all TCP traffic, etc.). Each element holds an incremental sequence number of the most recent packet seen to belong to the specific flow, and a timestamp indicating the arrival time of this packet. At any given point, the temporal difference from the arrival time of the most recent packet, indicates the inactivity timer of the corresponding flow. The flow table has a fixed size of one thousand entries upon exhaustion of which any new entry replaces the oldest existing entry (with the highest inactivity timer). It is reasonable to assume that the flow table size can mostly accommodate the number of flows running in parallel at an end-system. At the same time, the memory occupancy of the flow table is only 45 KB. An asynchronous periodic process examines the inactivity timer of each entry of the flow table, and removes entries that have been inactive for longer than a given threshold value. This process prevents unnecessary space being occupied by the flow table, and also implicitly facilitates fast entry retrieval by disassociating inactive entries. Keeping flow state at the network layer may

seem to be an overhead component; however, it facilitates an opaque and independent network-layer packet loss implementation, equally applicable to all current and future transport and application layer protocols (e.g. TCP, UDP, etc.). It also provides the flexibility of defining flows of different granularity according to the measurement scope. Using existing transport protocol control blocks (e.g. the hash table of active sockets) to embed the necessary structures for packet loss measurement would make the overall operation less opaque, since modifications at different layers (and for different protocols) of the stack would be necessary. At the same time, packet loss measurement would be applicable only for individual microflows (and not flow aggregates) that would originate locally. The flow specification would not extend to flows routed through (but not originated at) a particular node.

The SRC module needs to avoid causing fragmentation when instrumenting traffic with in-line measurement indicators, since this would have a detrimental effect on the performance of instrumented packets which would be largely different from the rest of the traffic. This is less of an issue for UDP applications (e.g. VoIP) since they very rarely use large packet sizes. For bulk TCP traffic, however, which is carried within maximum-sized segments, the SRC module needs to communicate its space requirements to the stack. A TCP implementation computes the Maximum Segment Size (MSS) to transmit based on the minimum value between the MSS it receives from the remote end (in a SYN or SYN+ACK message), and its own medium's Maximum Transfer Unit (MTU) after subtracting the space reserved for the IP and TCP headers plus any IP options, if present [17]. The SRC in-line measurement module can communicate its space requirements by setting the *ext_header_len* variable in a connections' *tcp_opt* structure to the measurement extension header's size. However, this would require the measurement module to monitor the active connections' hash table and adjust the corresponding variables for connections that match the filtering criteria of the measurement process. In addition, when a socket is locally constructed in response to an application request, the local MSS value will be computed before any packet reaches the measurement instrumentation module. The most cost- effective way (which has been adopted by this implementation) to alter the MSS in order to accommodate space for the measurement headers is to monitor incoming SYN or SYN+ACK packets, and if they match the measurement filter specification, to clamp the advertised MSS (within the packet structure) of the remote end to *min(local_MSS, remote_MSS) – hdr_ext_len*. If, for any reason, (e.g. existing TCP sessions that have negotiated MSS prior to the initiation of a SRC measurement module) the addition of a measurement header would cause fragmentation, the SRC module will leave the packet unaltered.

A generic DST measurement module implements one main function to retrieve the measurement extension header and its operation is triggered by the presence of such header in an incoming IPv6 packet. For the unidirectional delay measurement, a system timestamp of packet arrival at the destination of an instrumented path is read and inserted in the corresponding fields of the measurement header, whereas for the packet loss measurement the contents of the header remain intact. The main purpose of the DST module is to capture the measurement extension header and to subsequently remove it from the packet. It is worth noting that removal of the measurement extension header is not compulsory but it is included for opaqueness. If the header is not removed by the intended destination of an instrumented path (or if e.g. a DST module is not running at the destination), the stack will simply ignore it and process the rest of the packet.

The module adjusts the packet structure's size (*skb_pull*) and updates certain fields (*next header, payload length*) of the immediately preceding protocol header, accordingly. The extracted header is inserted into a FIFO queue and can be read by higher-level measurement processes through a system call to the kernel module. The FIFO structure has a fixed length of ten thousand bytes and it therefore imposes an upper limit on the amount of memory consumed by in-line measurement headers extracted from arriving datagrams. A consuming application can retrieve per-packet measurement records from the queue, either one-at-a-time or in bulks. A read operation from a consuming application results in the corresponding number of bytes being freed from the queue. When the queue if full, extension headers from newly arriving datagrams are not stored until space has been freed. It is left to the application to determine the appropriate pace to consume measurement data (depending also on the temporal traffic rate) in order to maintain an appropriate queue length and avoid newly arriving data being dropped locally. The chosen size for the data structure allowed user space applications to consume results continuously without losing any packet information.

IV. MEASUREMENT DATA OVERHEAD

The overhead associated with the operation of network measurement mechanisms is typically judged with respect to the measurement process and the measurement data injected into the network. In-line measurement does not generate additional synthetic load between two instrumented points in the network. Rather, measurement indicators are inserted in the data-carrying traffic itself which creates a small and constant data overhead of either 8 or 24 bytes per-packet for measuring packet loss and delay, respectively, and it is irrespective of the traffic type, rate or any other characteristics. This accounts for a byte overhead of 0.5% when measuring packet loss of maximum-sized Ethernet segments (1518 bytes) and can get up to 23% when measuring the delay of minimum-sized TCP/IPv6 acknowledgment packets. Although the latter percentage may seem quite significant, it is worth pointing out that acknowledgments consist entirely of headers and hence the addition of the measurement extension does not really cause data traffic reduction. In addition, it is envisaged that measurement instrumentation will mainly be targeted at the

larger and maximum-sized packets that carry the actual user data, and minimum-sized acknowledgement packets will only occasionally and very selectively be instrumented when particular aspects of this type of traffic will be considered.

Measuring the same properties using an active measurement mechanism would require the generation of synthetic load of identical characteristics, volume and duration to the traffic to be measured. Hence, letting alone the bandwidth consumed by active measurement processes and the consequential side-effects on network performance degradation, they also impose linearly increasing data overhead with respect to the target traffic characteristics. In terms of bytes, the generation of an additional packet results in a link-layer byte overhead from 64 to 1518 bytes (for e.g. an Ethernet segment), which is a deficiency ranging from 63% to 99.48% when compared to in-line measurement. At the same time, the direct per-packet in-line measurement conducted at the destination of an instrumented path does not require offline measurement data to be shipped and correlated over the network, as it is the case with passive monitoring systems. The data correlation required from two distinct passive monitoring devices in order to measure a characteristic of the interconnecting path (such as the unidirectional delay) equals twice the amount of the per-packet captured data plus the capture library header that normally stores measurement data such as timestamps [5]. This would at least sum up to 2x56 bytes per packet for IPv4 and 2x76 bytes per packet for IPv6 (to include network and transport layer headers and e.g. the *libpcap* header), which results in a data overhead deficiency between 78.6% and 94.7% compared to in-line measurement.

V. END-TO-END THROUGHPUT OVERHEAD ASSESSMENT

We now turn our attention to a rigorous experimental evaluation of a software-based end-to-end in-line performance measurement implementation with respect to the overhead incurred by the measurement process itself.

A. Experimental Environment and Parameters

The experimental environment consisted of two Sun Fire X4100 servers equipped with one 2.2 GHz AMD Opteron™ processor (single-core), with 128 KB of L1 cache and 1MB of L2 cache. The machines have 2 GB of 400 MHz DDR synchronous memory. The network interfaces are Gigabit Ethernet Intel® PRO/1000 residing in a PCI-X 100 MHz slot. The two systems were connected via a Force10 E1200 switch/router configured as an IPv6 router that didn't drop any packets during the experiments. The end-systems ran Linux 2.6.20, (re)compiled to support the in-line measurement hooks, and had the corresponding SRC and DST kernel modules loaded to perform end-to-end in-line unidirectional delay and packet loss measurement, respectively.

An application-level traffic generator has been used to operate on top of the socket layer to resemble the packet generation patterns and performance of application flows.

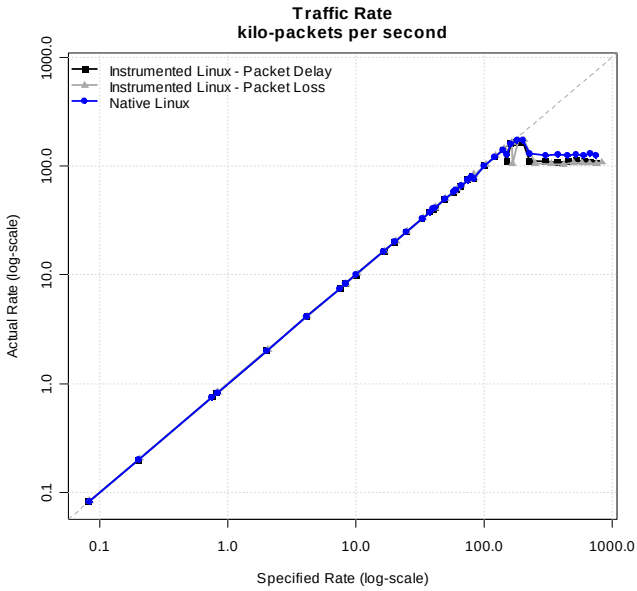

Fig. 4. Actual vs. specified traffic rate for instrumented and native Linux

Iperf was used to generate a range of CBR UDP/IPv6 flows assuming different packet sizes and different transmission rates [18]. The minimum packet size supported by Iperf in order to stream its own measurement indicators between the client and the server processes is 56 bytes of application-level data. This results in a minimum Iperf/UDP/IPv6 datagram of 104 bytes, and a 112-byte or 128-byte minimum-sized datagram for one-way loss and one-way delay in-line measurement instrumentation, respectively. We have therefore chosen 56, 512 and 1400-byte *application-level* packets to represent minimum, medium and maximum-sized IPv6 datagrams. Traffic rates were varied between 1 Mb/s and 1 Gb/s, which is the typical maximum transfer rate for commodity hardware/software end-system configurations. For each combination of transmission rate and packet size, the experiments were replicated three times. Wildcard filtering and sampling specifications were used for the SRC delay and loss modules to instrument all outgoing traffic.

B. Instrumentation impact on end-to-end throughput

We have studied the impact of traffic instrumentation on end-to-end application-level throughput due to the additional per-packet operations incurred. The nested combination of different packet sizes and transmission rates used for the experiments has been normalised to kilo-packets per second, using the decimal conversion for high orders of magnitude of transmission speeds. Fig. 4 shows the effective end-to-end rate achieved (in kilo-packets per second) versus the rate specified by the traffic generation process for the native and instrumented Linux, respectively. Each point in the graph represents the mean packet rate taken from the three experiment replications for a given set of parameters. For most packet rates, the overhead incurred from the SRC and DST modules for both the packet delay and packet loss measurement instrumentation is negligible since the specified

rate is sustained. For the highest rates, from 180 up to the theoretical maximum of 880 kilo-packets per second (for 56-byte, application-level packets), the system operates under stress and this is evident from the deviation of the effective rate from the specified rate for native Linux. These traffic rates correspond to minimum-sized (56-byte) packets transmitted at 200 Mb/s up to 1Gb/s. Mean throughput reduction of 3.4% and 6% for packet delay and loss instrumentation, respectively, only occurs at these high-load rates, when even native Linux's effective throughput is reduced by 15.3% on average. The additional flow classification and table lookup operations undertaken by the loss SRC module make it more costly on average than the stateless packet delay measurement. It is visually evident that the use of minimum-sized packets at high transmission rates has a far more detrimental effect on the end-to-end throughput than in-kernel measurement instrumentation. In addition, this is the worst-case performance bound incurred by in-line measurement instrumentation, and it can be seen that by employing a moderate systematic sampling scheme (e.g. instrumenting 1-in-10 packets), measurement cost on throughput can drop below 1%. This becomes clearer by comparing the maximum effective application throughput for each of the three different packet sizes, as shown in TABLE I. Using 1400-byte packets, native Linux throughput approximates the maximum application-level theoretical transmission rate for a system equipped with 1Gb/s interfaces, due to per-packet header overheads [8]. The difference between the maximum effective throughput achieved by native and instrumented Linux is minimal for maximum and medium-sized packets, varying between 1.5% and 3.4% for unidirectional delay, and between 0.6% and 1.3% for packet loss instrumentation, respectively. On the other hand, throughput of minimum-sized packets is massively decreased even for native Linux, reaching only 181 Mb/s. However, this throughput reduction does not reflect the modules' overhead; rather, it is caused by the increased context switch between user and kernel address space when the end-systems operate under high load.

We have experimentally evaluated this claim by using the *pktgen* [19] in-kernel traffic generator to stress test the systems' maximum transmission rates. We modified the *pktgen* source code to include the in-line traffic instrumentation and compared its effective throughput with a clean *pktgen* instance. With 56-byte (application-level) packets, the one-way delay and the packet loss instrumented versions achieved 641 kpps (604Mb/s) and 619 kpps (584Mb/s), respectively. The original pktgen version reached 677 kpps (638Mb/s). These significantly higher throughput values verify the costly user space/kernel context switch.

TABLE I. MAXIMUM END-TO-END APPLICATION THROUGHPUT

Traffic instrumentation	Maximum application throughput by packet size (Mb/s)		
	56-byte	512-byte	1400-byte
Packet delay	154	822	922
Packet loss	134	840	930
Native Linux	181	851	936

The measurement modules' impact on throughput even under maximum system load (in terms of packet-per-second generation) remains below 6% and 9%, for packet delay and loss instrumentation, respectively. Although it is evident that a measurement process would tune itself to not instrument minimum-sized datagrams transmitted at maximum rate since they do not represent any typical end-to-end application-level load, this stress-test demonstrates the low overhead of in-line traffic instrumentation even under maximum system load. Again, employing a moderate systematic sampling scheme would result in measurement cost reduction below 1% under extreme load conditions.

C. Factorial design and analysis

After looking at the mean impact of the instrumentation process on end-to-end throughput, we will now assess the importance and statistical significance of these values, with respect to other influencing parameters, such as the traffic generation rate and the datagram size. For this purpose, we have constructed a full three-factor factorial experimental design with replications, and analysed the corresponding regression model. The response variable has been the ratio of the effective end-to-end throughput over the specified transmission rate of the traffic generation process. The three predictor variables that affect the response are the presence of the measurement modules (*A*), the transmission rate (*B*), and the packet size (*C*). The analysis will enable separating the effects of each factor on performance, and determining if a factor has a *significant* effect or if the observed differences in the response variable are due to *random variations* caused by uncontrolled experimental parameters. The model for the 3-factor full factorial design, with factors *A*, *B*, and C at *a*, *b*, and *c* levels, respectively and *r* replications is:

$$y_{ijkl} = \mu + \alpha_i + \beta_j + \xi_k + \gamma_{ABij} + \gamma_{ACik} + \gamma_{BCjk} + \gamma_{ABCijk} + e_{ijkl}$$
$$i = 1,...,a;\ \ j = 1,...,b;\ \ k = 1,...,c;\ \ l = 1,...,r \qquad (1)$$

The model includes the mean response μ, 2^3-1 effects and the experimental error; three main effects, three two-way factor interactions and one three-way factor interaction between all predictors. α_i is the effect of factor *A* at level *i*, γ_{ABij} is the interaction between factors *A* and *B* at levels *i* and *j*, respectively, and so on. In our case, the measurement module factor has two levels, indicating whether the SRC and DST modules are loaded or not. Transmission rates assume continuous values from 1 Mb/s up to 1 Gb/s, and the packet size factor has three levels for 56, 512, and 1400-byte packets, respectively. For each factor-level combination, the experiment was repeated three times. The importance of each factor is measured by the proportion of the total variation in the response that it explains. Its statistical significance is then calculated using the typical analysis of variance procedure which compares the contribution of the factor to variation with respect to that of the unexplained variation due to errors [20].

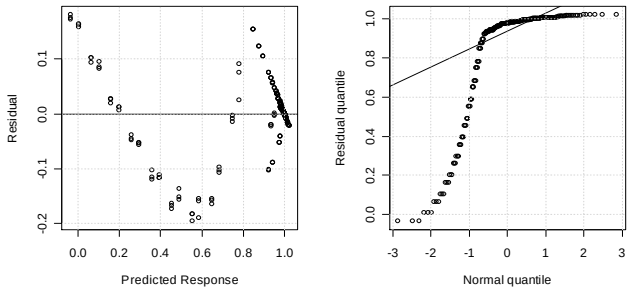

Fig. 5. Residuals versus predicted response and normal quantile-quantile plot of the residuals

However, the underlying assumptions in deriving the expressions for model effects are that the model errors are statistically independent, additive, normally distributed and have a constant standard deviation σ_e, and that the effects of factors are additive [21]. These assumptions lead to the observations being independent and normally distributed with constant variance. Fig. 5 shows that the assumptions of normality and independence of errors do not hold for our regression models. The scatter plot of residuals (model errors) versus predicted response show clear trends that cannot be ignored since the two quantities lie in the same order of magnitude. Also, the normal quantile-quantile plots of the residuals show that they heavily deviate from normality. As it was expected, obvious transformations of the models' response variable did not satisfy the normality and independence assumptions either, mainly due to its skewness (the ratio of the effective versus specified throughput for the majority of experiments is close to or equal to 1). Therefore, in order to robustly measure the importance and significance of each factor we have used *nonparametric bootstrap* to estimate the sampling distribution of the model effects and their sums of squares, without making any assumptions about the form of the population and without deriving the sampling distribution explicitly [22]. For each of the two types of traffic instrumentation, the set $S = \{X_1, X_2, ..., X_n\}$ of the effective throughput ratio of all runs is considered as a sample from the population $P = \{x_1, x_2, ..., x_N\}$ of all possible outcomes.

Nonparametric bootstrap draws a sample of size n with replacement from among the elements of S to form the resulting bootstrap sample $S_1^* = \{X_{11}^*, X_{12}^*, ..., X_{1n}^*\}$. In effect, the sample S is treated as an estimate of the population P, where each element X_i of S is selected for the bootstrap sample with probability *1/n*. This process is mimicking the original selection of the sample S from the population P, and is repeated a large number of times selecting many bootstrap samples. The key bootstrap analogy is that the population is to the sample as the sample is to the bootstrap samples. Consequently, a statistic $T_b^* = t(S_b^*)$ computed for each of the bootstrap samples has a distribution around the original estimator $T = t(S)$ of the sample analogous to the sampling distribution of T around the population parameter $\theta = t(P)$. One thousand bootstrap samples were used to estimate the distributions of the sums of squares and of all the model effects.

TABLE II. shows the percentage of variation allocated to factors, interactions and errors. For both packet delay and packet loss measurement instrumentation, the overall model explains more than 94% percent of the variation in end-to-end effective throughput ratio, since the variation due to errors is less than 6%. In both cases, the measurement modules' contribution to variation is minimal, accounting for less than 0.6%, and can therefore be safely ignored. Transmission rate explains 18.05% of the variation in effective throughput for the packet delay instrumentation, and 19.96% for the packet loss instrumentation. Clearly, the most important parameter is packet size that explains more than 50% of the variation in both models. The interaction between transmission rate and packet size for both models explains 24.09% and 23.43% of the variation, respectively. On the contrary, the two-way interactions between the measurement modules and the transmission rate and packet size are unimportant since they account for less than 1% of the variation in both models. Likewise the three-way interaction between the predictors can also be safely ignored for both models. Overall, the allocation of variation shows that the presence of the measurement modules does not impact the applications' end-to-end effective throughput.

Fig. 6 and Fig. 7 show all the bootstrapped model coefficients with their 95% bias-corrected, accelerated (BC_a) percentile confidence intervals for the packet delay and packet loss instrumentation models, respectively. BC_a improves on the percentile method – which also does not assume normality – by correcting bias and skewness [22]. The figures show the contrasts of the *k-1* levels of each factor, omitting the base level. For the measurement modules, the base level is their absence from the traffic generation process, and for the packet size factor is the 1400-byte value. All *k* levels for each factor sum to zero. The coefficients that have a statistically significant effect on the effective throughput ratio, at a 0.05 significance level, deviate from the zero reference line. For example, if we look at Fig. 6 it is evident that the choice of a small packet size (56 bytes) has a significant effect in decreasing the effective throughput due to the large number of packets that need to be generated to sustain the highest bandwidth levels.

TABLE II. ALLOCATION OF VARIATION OF FACTORS (%)

Allocation of variation (%)				
Traffic instrumentation	Factors			
	Module	Rate	P_size	Errors
Packet Delay	0.46	18.05	51.67	5.19
Packet Loss	0.59	16.96	52.37	5.87
	Interactions			
	mod/rate	mod/p_size	rate/p_size	3-factor
Packet Delay	0.16	0.31	24.09	0.06
Packet Loss	0.16	0.56	23.43	0.06

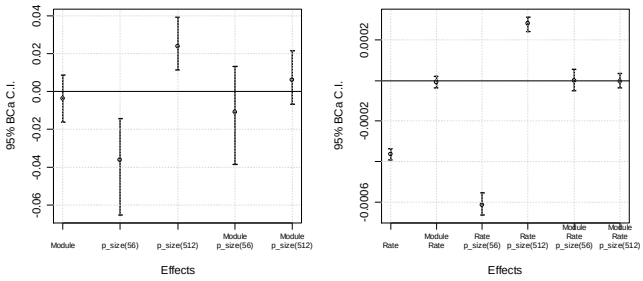

Fig. 6. Unidirectional delay instrumentation: model effects with 95% *BCa* Confidence Intervals

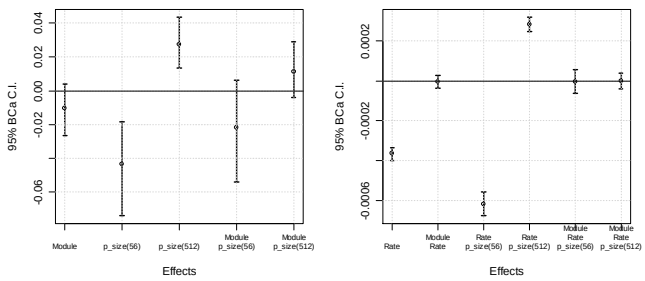

Fig. 7. Packet loss instrumentation: model effects with 95% *BCa* Confidence Intervals

To a lesser extent, increasing transmission rate has a similar effect (rightmost plot) since the system does not have the internal capacity to generate the highest bit rates (approaching 1 Gb/s) irrespective of packet size. It is worth noting here that maximum network capacity is defined with respect to the link-layer transmission capabilities, and it is therefore normal for the application to never reach the maximum nominal bandwidth since headers from lower levels cause additional byte overhead. Both Fig. 6 and Fig. 7 agree on the statistically significant effect that transmission rate and minimum packet size have on decreasing the effective end-to-end throughput. The interaction between transmission rate and packet size also has a statistically significant effect, especially for minimum-sized packets. On the contrary, the presence of the measurement modules does not impact end-to-end throughput in a statistically significant way at a 0.05 significance level, since the 95% confidence intervals of the coefficient for both models include zero. In addition, all the interactions between the measurement modules and the rest of the factors are insignificant at the 0.05 confidence level.

Therefore, after this rigorous experimental design analysis, it is safe to conclude that the per-packet, in-line measurement traffic instrumentation does not negatively impact end-to-end effective throughput in a statistically significant way.

VI. SYSTEM PROCESSING OVERHEAD

A. System-wide instrumentation impact

After demonstrating that in-line traffic instrumentation does not have a statistically significant impact on application-perceived performance, this section focuses on the system processing impact of the in-line measurement modules at the two instrumented end-systems that insert and record/remove the measurement extension header, respectively. We have used the *oprofile* tool to examine the modules' system-wide processing impact on the source and destination end-systems [23]. Oprofile is capable of profiling all running code of a system, including interrupt handlers, the kernel, shared libraries and applications by leveraging hardware registers of the CPU that provide performance counters for cache misses, CPU cycles, etc. We have measured the mean CPU utilisation of all running code on the two instrumented end-systems during the traffic generation process with and without the modules being loaded, and compared the differences. Due to profiling granularity restrictions, we have accounted for functions individually consuming at least 0.001% of CPU time, and we then normalised them to sum to 100%. For each function, the average CPU utilisation over three replications of the experiments was calculated, and then the distribution of per-function CPU utilisation was computed. By computing the differences in the utilisation distributions between the instrumented and native end-systems, we calculated the mean difference in per-function CPU utilisation and computed the 95% quantile confidence intervals, which are shown in Fig. 8 and Fig. 9. It can be seen that for both types of in-line traffic instrumentation and at all packet transmission rates, the difference in CPU utilisation at a 0.05 significance level for both the source and the destination end-systems includes zero and is therefore statistically insignificant. More specifically, it can be seen that the confidence intervals cover a minimal region of less than 4% and therefore the variability in CPU consumption by the various systems processes is minimum. This implies that the presence of the SRC and DST modules on systems' protocol stacks for either unidirectional delay or packet loss instrumentation does not incur a significant system-wide processing overhead by not causing any system process (like e.g. kernel socket buffer functions) to significantly increase its CPU consumption, even when instrumenting traffic at high packet rates.

TABLE III shows the mean CPU utilisation (%) of each function of the delay and loss measurement modules. As it will become apparent in the following subsection, individual utilisation values should be treated as relative estimates, since the profiler would not accumulate the utilisation of external function calls to the caller function. However, it is useful to compare the differences in the overall CPU utilisation between the two types of measurement instrumentation.

TABLE III. MEAN PER-FUNCTION CPU UTILISATION (%)

	Mean CPU Utilisation (%)				
	filter_tests	sampling	make_room	insert	retrieve
Delay measurement	0.83	0.11	0.26	0.33	4.27
	1.53				
Loss measurement	1.12	0.19	0.21	0.54	2.52
	2.06				

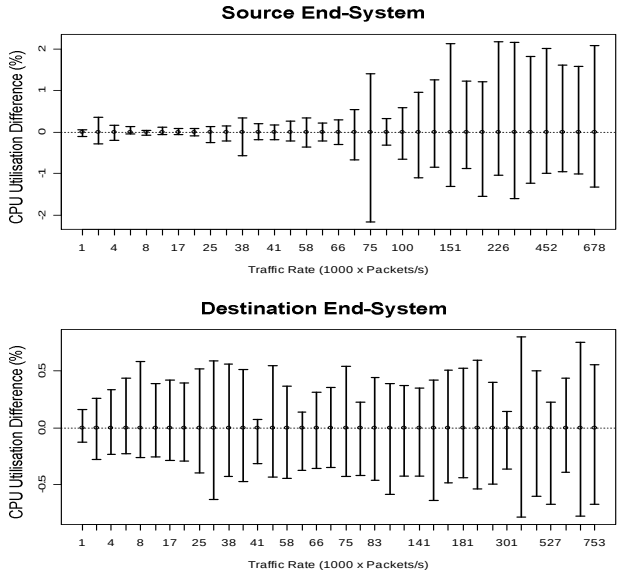

Fig. 8. Unidirectional delay instrumentation: impact on system-wide CPU utilisation

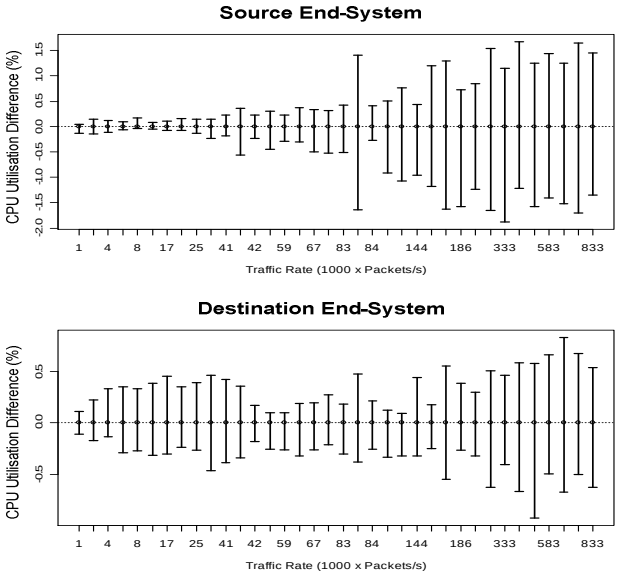

Fig. 9. Packet loss instrumentation: impact on system-wide CPU utilization

The SRC module incurs less CPU overhead for the packet delay instrumentation than it does for packet loss, due to the internal (flow) state that is maintained by the latter. In order to insert the appropriate sequence number to each datagram, the SRC module needs to match it against a given flow specification held in memory, and to subsequently update this specification to include the latest reference values. This is clearly more costly than the stateless per-packet operation of reading one system timestamp, undertaken by the one-way delay SRC module. On the contrary, the DST module undertakes a more costly operation when performing a delay measurement, since it needs to amend the measurement header with an additional system-local indicator. All other operations being equal (e.g. extension header extraction, amendment of the socket buffer structure, etc.), the packet loss DST module merely stores the existing header without modifying it.

B. Detailed kernel instrumentation

Oprofile is an external CPU sampling utility and it can provide for system-wide percentage profiling, but cannot accurately assess the exact CPU utilisation of individual functions. This is due to the external profiler sampling the innermost function and therefore utilisation is not attributed to the caller functions. Therefore, in order to get absolute values of CPU usage for the in-line measurement modules, we have used the ReaD Time Stamp Counter (RDTSC) CPU instruction to instrument every single function of the modules' code and compute its processing cost. During each experimental run, we recorded the processing time for each instrumentation function of the last one thousand packets. Most (per-packet) processing times assumed values around a single mode with a few observations deviating and creating skewness in the distribution, hence the median was chosen as the index of central tendency for the per-function processing time distributions in each experimental run [21]. Then the mean processing time over the three replications of each experiment was chosen for each function.

Fig. 10 and Fig. 11 show the three-dimensional scatter plots of the per-packet processing time in CPU cycles versus traffic rate and packet size for each instrumentation function of the delay and loss-measurement modules, respectively. In each scatter plot, a plane is also drawn based on the linear model of the processing time with respect to the transmission rate and the packet size. The plane helps to visually identify the dependence of the response variable to the two altering factors. The first four functions are employed by the SRC module and the latter by the DST module. For both types of instrumentation, packet filtering and header insertion functions assume small values independently of packet size and transmission rate. This was expected since both functions perform invariable operations based on the header contents of each packet. So does packet sampling, although its operation is minimal since it was set to sample all packets for the present experiments. It can be seen that header insertion is more than twice as costly for the stateful packet loss instrumentation as it is for packet delay. This is due to the additional flow classification and lookup operations are conducted by the packet loss SRC module, in order to determine the correct sequence number to insert to the in-line measurement header. The most costly function is make_room_for_ext_hdr that invokes the corresponding system calls to increase the socket buffer's headroom and to push the packet contents accordingly [16]. These heavily depend on the packet structure's size since larger data blocks need to be moved in the kernel; to some extent the cross-function calls also increase consumption under high CPU load. This important attribute is not captured by the external profiler's values in TABLE III which do, however, demonstrate the overall relative cost difference between the SRC and DST modules of the two types of instrumentation.

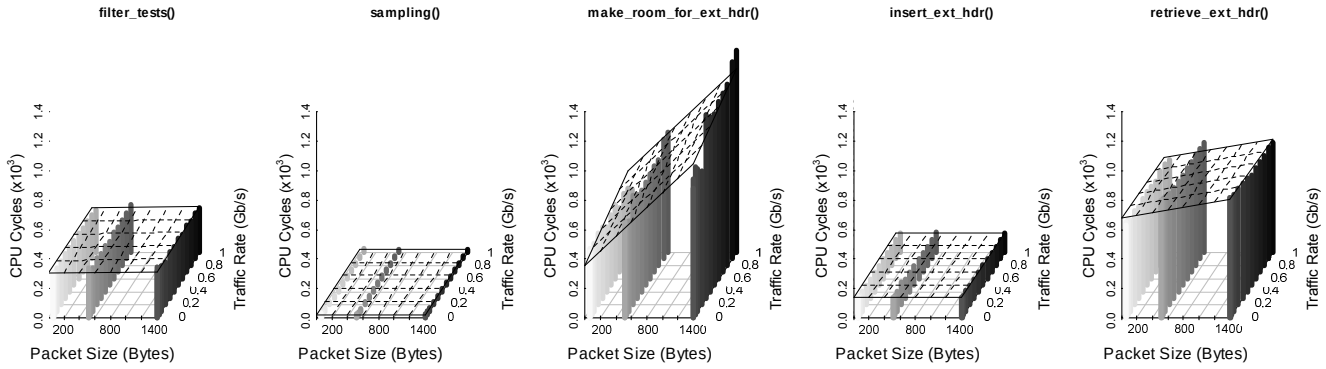

Fig. 10. Unidirectional delay instrumentation: CPU Cycles consumed by the SRC and DST modules' functions

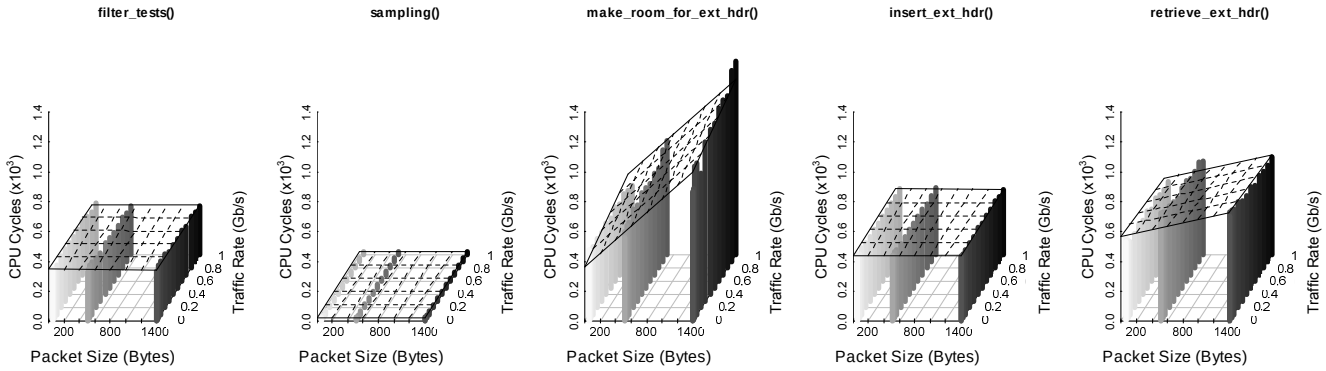

Fig. 11. Packet loss instrumentation: CPU Cycles consumed by the SRC and DST modules' functions

Retrieve_ext_hdr is the second most costly function whose CPU cycles consumption mainly depends on the packet size. This is due to the removal of the measurement extension header and the corresponding adjustment of the packet structure's headroom (inverse operation from that undertaken by the SRC module's make_room_for_ext_hdr), and also due to copying each extracted header to a memory queue which can then be read by higher-level applications. It is up to these consumer applications to decide on the method and the temporal interval at which they should read the extracted headers. The instrumentation modules take care not to exhaust memory resources by maintaining a fixed-size structure, after exhaustion of which (due to, for example, a consumer process reading data only infrequently) no further measurement extension headers are stored. The slightly higher CPU consumption of this function for the unidirectional delay measurement, relates to the successive memory copies of the larger (delay) measurement extension header.

TABLE IV shows the overall processing time of in-line end-to-end packet instrumentation for the two types of measurements in microseconds, on the 2.2 GHz CPU of our instrumented end-systems. Overall processing time is the sum of all SRC and DST modules' functions for each type of measurement. It can be seen that the total processing time of the software-only in-line measurement instrumentation is minimal, on the order of one microsecond.

TABLE IV. OVERALL TRAFFIC INSTRUMENTATION TIME BY APPLICATION-LEVEL PACKET SIZE ON 2.2 GHZ CPU

Total Instrumentation Time (μsec)			
	56-byte	512-byte	1400-byte
Unidirectional Delay	0.71	0.88	1.08
Packet Loss	0.81	0.97	1.16

Moreover, the two most costly functions are dominated by memory access operations (such as e.g. *skb_cow*, *skb_pull*, etc.), implying that the CPU has to spend idling cycles if the data were not in its cache. Hence, the overall 'true' processing time can be further reduced by exploiting data locality during packet processing.

The impact of increasing packet size on the processing time is minimal, since on average, 1400-byte packets only incur a 0.36 μsec additional overhead with respect to minimum-sized packets. The total stateless packet delay instrumentation is on average 0.09 μsec less costly than packet loss instrumentation which maintains internal (flow) state, regardless of packet size.

VII. DISCUSSION

In-line measurement is a mechanism to measure the performance of the data-carrying Internet traffic while this is routed between a source and a destination (either the ultimate end-points or intermediate ones, between for example, network

ingress and egress). It merges the benefits of active and passive measurement into what can be seen as a hybrid: directly implementing a chosen performance metric in short timescales (similar to active), by observing and instrumenting the existing operational traffic (similar to passive). At the same time, it overcomes the major limitations of active and passive measurement: first, it avoids the "Heisenberg" effect of active measurement where the additional traffic perturbs the network and biases the resulting analysis; it also avoids the need for correlation and analysis of passive measurement traces, and the consequential need to operate in long timescales. The main challenge for such in-line measurement mechanism is to have a small impact on the network traffic and on the instrumented systems, so that it can form the basis for a measurement plane for the next generation Internet and operate at an always-on manner. In this paper, we have focused on the end-to-end software implementation of in-line measurement, and we showed that an in-kernel prototype system can be seamlessly integrated with the network stack and incur minimal overhead. In a different study, we have built a system for instrumenting traffic with in-line measurement headers between the edges of network topologies [15]. In that case, hardware is exploited in order to keep pace with the multi-gigabit rates, and to facilitate in-line measurement instrumentation as a native part of routers' data (fast) path.

In its current form, in-line measurement has been designed as a two-point mechanism in order not to incur significant additional processing to the data-carrying traffic, and to demonstrate how the measurement functionality needs only to occur at specific pre-identified nodes which can be adequately provisioned. However, the mechanism can be easily extended to implement different performance metrics such as hop-by-hop or round-trip delay and loss. The appropriate header structures would need to be defined in order to carry the relevant indicators for the desired metrics. For example, for measuring intermediate path delays between multiple network nodes, header fields would need to hold the relevant number of timestamps along with the corresponding node identifiers. However, increasing the number of instrumentation points along a path reduces selective processing and introduces additional overhead on the instrumented traffic which when accumulated can be non-negligible. Therefore, the associated overhead should be carefully considered when implementing performance metrics that require processing from intermediate network nodes.

One important property of in-line measurement is that the mechanism is independent from particular measurement processes and/or infrastructures. Therefore, it can be integrated with higher-level processes developed to measure specific properties of a path. For example, a process to measure capacity or available bandwidth of a path can generate its own traffic load (e.g. trains of packet-pairs) and use the in-line measurement headers to measure packet inter-spacing at source and destination, taking advantage of the kernel-level timestamps. Using the filter specification of the in-line measurement prototype, an application process can choose a particular subset of traffic to be instrumented by the kernel. The seamless integration of in-line measurement with the network stack, its potential always-on operation, together with its different instantiations (end-to-end, edge-to-edge), constitute it a promising candidate mechanism for network operations and management. It provides a unified and extensible framework able to instrument any type of traffic over any type of network, and produce accurate results that reflect the temporal performance experienced by the operational network load.

Although measurement functionality has been incorporated in a number of protocols, particularly those handling real-time traffic such as RTP, in-line measurement can become a universal mechanism for the measurement and management of all traffic carried over the next generation Internet infrastructure. In contrast to transport and application-level measurement which can be deployed end-to-end and used only by the relevant end-systems, in-line measurement can be equally exploited by end-systems and by network operators. At the same time, it can be exploited by any new protocol deployment, which will not need to build its own redundant instrumentation mechanisms to tune the application-level end-to-end performance.

End-to-end deployment of in-line measurement on individual microflows and flow aggregates can provide an accurate description of the service levels delivered to customers, and enable performance-based charging, especially for intolerant applications with real-time requirements. Summaries of temporal performance indicators can be exchanged between the end-system and the service provider to give timely views of user-perceived performance, and potentially enable the last-mile topologies to be provisioned on-demand according to the application requirements and the service-level agreement. Similar, yet totally static, approaches on service differentiation are being used by some providers who give the ability to users to flex their network speed for a given time interval. By using in-line measurement data available at end-systems, providers can offer more fine-grained service differentiation based on ephemeral needs of particular application flows.

An in-network edge-to-edge deployment can also be of particular relevance to operators for directly measuring the performance of their topologies, and not having to infer it through correlating sampled traces at much longer timescales. Large objects such as traffic matrices and also edge-to-edge delay and loss can be directly computed for different traffic types and at different levels of aggregation. At the same time, since measurement is based on the actual data-carrying traffic, an operator can integrate performance indicators with network control structures that can enable, for example, load-sensitive routing and traffic differentiation. Distributed measurement infrastructures can aggregate and consume measurement indicators from instrumented nodes, and can construct network-wide views of performance in short timescales.

VIII. RELATED WORK

Active and passive measurement mechanisms do not typically enhance the packet forwarding functions of the data-carrying traffic. Active measurement infrastructures and tools (e.g. [3][4][18][19]) operate on synthetic traffic, whereas passive measurement systems observe the operational network traffic (e.g. [5][6][7]). Particular implementations, mainly of active measurement, have been designed to operate in the kernel in order to provide for increased control and efficiency over the measurement process yet they still do not interfere with the protocol stack operations of the data-carrying packets. Such tools include MAD [24] which is a kernel-level daemon to support real-time scheduling of probe streams, pktgen [19] which is a high performance traffic generator included in the Linux kernel, PeriScope [25] which is a kernel-level API enabling the definition of new probing structures, and pktd [26] which is a kernel daemon used to provide controlled access to the network device for higher-level measurement software. Reports of these tools do not provide detailed results of system impact since this varies depending on the different measurement processes used alongside.

A number of in-line measurement mechanisms have more recently been designed to offer measurement capabilities at different locations of the networking stack. Inline Measurement TCP (ImTCP) [27] and the Measurement Manager Protocol (MGRP) [28] operate at the TCP layer of the network stack and multiplex measurement and application traffic in order to infer network bandwidth. The former alters the TCP sending process to measure available bandwidth, whereas the latter uses measurement traffic to piggyback application data. In contrast to in-line measurement described in this paper, these tools implement a particular performance metric and do not constitute a wider framework for performance measurement. The evaluation of each system mainly focuses on the accuracy of the bandwidth measurement process and not so much on their overall system impact, which still needs to be analysed. Sidecar [29] suggests re-using retransmitted TCP segments to probe the network, and BitProbes [30] proposes to insert measurement information within the application payload of packets. Again, both papers focus on the relevance of their measurement results and not on the system-wide impact of the implementations.

Our solution is more generic than the propositions described above. Every packet can be tagged without any application or transport layer dependency. Consequently, it becomes easier to implement as it only requires the presence of an extensible optional framework, which is already natively present within IPv6 in the form of extension headers. Nevertheless, on the host side, there could be significant overhead incurred, since the measurement instrumentation can happen on virtually every IP packet. In this work, we have used fine-grained system performance analysis to demonstrate that this is not the case even if all data-carrying packets are tagged back-to-back. The work in [31] focuses on assessing the overall system impact of the TCP/IP stack implementation and uses techniques of host profiling similar to the one that we have used in this work. More recently, system-level performance evaluation of network protocol processing has been conducted on top of multicore systems, but focused on the hardware performance hit albeit without evaluating any protocol in particular [32].

IX. CONCLUSION

Following the legacy of telecommunication networks, the Internet has adopted a clear separation between control and data plane operations. At the same time, 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 and traffic performance evaluation non-trivial in short timescales. Performance measurement tends to be an ad hoc activity that is conducted independently of the network's main forwarding mechanism. Inevitably, most research focuses on performance modelling for the characterisation of traffic behaviour and for network provisioning. Ubiquitous mechanisms for instrumenting the data-carrying traffic, and therefore enabling for a unified framework for direct and pervasive performance measurement, have not been seriously considered because of the (often over-estimated) associated overhead on the Internet's data delivery mechanism.

However, the Next Generation Internet Protocol (IPv6) provides the necessary mechanisms that can be exploited for optional structures to be defined and encoded natively, as part of the ubiquitous network layer. In-line traffic instrumentation with measurement (and possibly control) information has a tremendous potential to become the cornerstone for automated Next Generation Networks (NGN) operations. The particularly low associated overhead, as demonstrated in this paper, should act as a driving force for future protocol designers to seriously consider the introduction of 'thin layers' for optional processing within the protocol stacks, in order to accommodate change and extensibility in NGNs.

In this paper we have thoroughly evaluated the impact of the software prototype of IPv6-based in-line measurement on throughput and on end-system resource consumption. Through rigorous and formal statistical analysis, we have demonstrated that an always-on traffic instrumentation mechanism can be seamlessly integrated with the network's main forwarding operation, while incurring minimal and statistically insignificant overhead. When operating under extreme load conditions, effective throughput reduction does not exceed 9%, whereas when transmitting at gigabit speeds with maximum-sized datagrams, end-to-end throughput reduction stays below 1.5%. These are worst-case figures when instrumenting every packet with measurement indicators. It is evident that by employing a moderate sampling scheme which will maintain measurement accuracy, the throughput reduction can be truly negligible. Per-packet CPU utilisation of the overall end-to-end traffic instrumentation process stays on the

order of one microsecond at a commodity 2.2 GHz processor, while the system-wide impact of the measurement modules on resource consumption is negligible.

REFERENCES

[1] Autonomic Network Architecture (ANA) Project, http://www.ana-project.org

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

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

[4] M. Alves, L. Corsello, D. Karrenberg, C. Ögüt, M. Santcroos, R. Sojka, H. Uijterwaal, R. Wilhelm, New Measurements with the RIPE NCC Test Traffic Measurements Setup, Passive and Active Measurement Workshop (PAM2002), USA, March 25-26, 2002

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

[6] D. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, C. Diot. Analysis of measured single-hop delay from an operational backbone network, IEEE INFOCOM'02, New York, June 2002

[7] N. Brownlee, Using NeTraMet for Production Traffic Measurement, IFIP/IEEE International Symposium on Integrated Network Management (IM'01), Seattle, USA, May 14-18, 2001

[8] 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

[9] D. P. Pezaros, D. Hutchison, F. Garcia, R. Gardner, J. Sventek, In-line Service Measurements: An IPv6-based Framework for Traffic Evaluation and Network Operations, IEEE/IFIP NOMS'04, April 2004

[10] A. Bianco, R. Birke, D. Bolognesi, J. M. Finochietto, G. Galante, M. Mellia, M. L. N. P. P. Prashant, F. Neri, Click vs. Linux: Two Efficient Open-Source IP Network Stacks for Software Routers, IEEE Workshop on High Performance Switching and Routing (HPSR'05), Hong Kong, May 12-14, 2005

[11] R. Bolla, R. Bruschi, Linux Software Router: Data Plane Optimization and Performance Evaluation, Journal of Networks (JNW), Academy Publisher, vol. 2, no. 3, pp. 6-11, 2007

[12] M. Luckie, A. McGregor, IPMP: IP Measurement Protocol, Passive and Active Measurement Workshop (PAM2002), USA, March 25-26, 2002

[13] D. L. Mills, S. Venters, Timestamp Capture Principles, on-line resource, 2009: http://www.eecis.udel.edu/~mills/stamp.html

[14] S. Deering, R. Hinden, Internet Protocol, version 6 (IPv6) Specification, IETF Network Working Group, RFC 2460, December 1998

[15] D. P. Pezaros, K. Georgopoulos, D. Hutchison, High-speed, in-band performance measurement instrumentation for next generation IP networks, Computer Networks (2010), Elsevier, doi:10.1016/j.comnet.2010.06.014

[16] K. Wehrle, F. Pählke, H. Ritter, D. Müller, M. Bechler, The Linux Networking Arhitecture, Pearson Prentice Hall, New Jersey, 2005

[17] R. Braden, RFC1122, Requirements for Internet Hosts – Communication Layers, IETF Standard, Network Working Group, October 1989

[18] Iperf – The TCP/UDP Banwidth Measurement Tool, On-line Resource, available at: http://dast.nlanr.net/Projects/Iperf/

[19] R. Olsson, pktgen the linux packet generator, The Linux Symposium, Ottawa, Canada , July 20-23, 2005

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

[21] R. Jain, The Art of Computer Systems Performance Analysis, John Wiley and Sons, Inc., ISBN 0471503363, 1991

[22] B. Efron, R. Tibshirani, An introduction to the bootstrap, New York: Chapman and Hall, ISBN 0412042312, 1993

[23] OProfile, a system-wide profiler for Linux systems, on-line resource, available at: http://oprofile.sourceforge.net/

[24] J. Sommers, P. Barford, An Active Measurement System for Shared Environments, -Internet Measurement Conference (IMC'07), San Diego, USA, October 24-26, 2007

[25] K. Harfoush, A. Bestavros, J. Byers, PeriScope: An Active Measurement API, Passive and Active Measurement Workshop (PAM2002), USA, March 25-26, 2002

[26] J. M. González, V. Paxson, pktd: A Packet Capture and Injection Daemon, Passive and Active Measurement Workshop (PAM2003), La Jolla, California, April 6-8, 2003

[27] T. Tsugawa, G. Hasegawa, M. Murata, Implementation and evaluation of an inline network measurement algorithm and its application to TCP-based service, in Proceedings of 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON 2006), Apr. 2006

[28] P. Papageorge, J. McCann, M. Hicks, Passive aggressive measurement with MGRP, ACM SIGCOMM'09, Barcelona, Spain, August 17-21, 2009

[29] R. Sherwood, N. Spring, Touring the Internet using TCP Sidecar, ACM SIGCOMM Internet Measurement Conference (IMC'06), Rio de Janeiro, Brazil, October 25-27, 2006

[30] T. Isdal, M. Piatek, A. Krishnamurthy, T.E. Anderson, Leveraging Bittorrent for End Host Measurements. Passive and Active Measurement Conference (PAM'07), Louvain-la-neuve, Belgium, April 5-6, 2007

[31] J. Kay, J. Pasquale, Profiling and Reducing Processing Overheads in TCP/IP, IEEE/ACM Transactions on Networking, 4(6):817-828, December 1996

[32] M. Faulkner, A. Brampton, S. Pink, Evaluating the Performance of Network Protocol Processing on Multi-core Systems, IEEE International Conference on Advanced Information Networking and Applications (AINA'09), Bradford, UK, May 26-29, 2009

Dimitrios P. Pezaros (M'00) is tenure-track Lecturer (Assistant Professor) at the School of Computing Science, University of Glasgow. Previously, he has worked as a postdoctoral and senior research associate on a number of UK Engineering and Physical Sciences Research Council (EPSRC) and EU-funded projects, on the areas of performance measurement and evaluation, network management, cross-layer optimisation, QoS analysis and modelling, and network resilience. He holds a B.Sc. (2000) and a Ph.D. (2005) in Computer Science from Lancaster University, and has been a doctoral fellow of Agilent Technologies (2000–2004). Dimitris is a member of the IEEE and the ACM.

Mickaël Hoerdt was awarded a PhD in Computer Science (Networking) from University of Strasbourg, LSIIT in 2005. His subject was multicast routing scalability and Internet topology mapping. He subsequently did one post-doc year at NTNU, Q2S-Trondheim, Norway, and three years as a research associate in the Vrouter project at Lancaster University, Infolab21, UK. He then worked as an invited researcher in the Ecode European project at the INL Networking Lab in Louvain-la Neuve, Belgium for six months. He is now a full time theology student at the Brussels Bible Institute in Belgium.

David Hutchison is Director of InfoLab21 and Professor of Computing at Lancaster University, and has worked in the areas of computer communications and networking for more than 25 years. He has recently focused his research efforts towards network resilience. He has completed many UK, European and industry-funded research contracts and published many papers as well as writing and editing books on these and related areas. He has been an expert evaluator and member or chair of various advisory boards and committees in the UK (EPSRC, DTI, OFTEL, e-Science, UKLight, UKCRC, JISC, DCKTN) and within the EU through several Framework Programmes. Also, he has served as member or chair of numerous TPCs (including the flagship ACM SIGCOMM and IEEE Infocom), and of journal editorial boards. He is an editor of the renowned Lecture Notes in Computer Science and of the Wiley CNDS book series