Evaluation of Building Blocks for Passive One-way-delay Measurements

Tanja Zseby, Sebastian Zander, Georg Carle

GMD FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
[zseby, zander, carle]@fokus.gmd.de

***Abstract*— Service Level Agreements (SLAs) specify the Quality of Services (QoS) negotiated between provider and customer. QoS Measurements provide a suitable way to proof the fulfillment of the given guarantees. During service usage the traffic of interest is already present in the network. This traffic can be utilized for passive (non-intrusive) measurement methods avoiding the disadvantages of sending test traffic for active (intrusive) measurements. Some applications (e.g. interactive applications like IP telephony) rely on guarantees for one-way metrics like one-way delay. One-way metrics usually cannot be derived from round trip measurements. Therefore specific methods are required to measure one-way metrics.**

In this paper we evaluate the necessary building blocks for passive one-way-delay measurement. In particular we discuss different functions for the generation of packet identifiers needed to correlate packet arrivals at the ingress and egress measurement point. We have implemented a flexible and modular meter, which allows us to use and compare different functions for the packet ID generation. In experiments we have measured the performance of the different methods investigated in this paper. We conclude with the experiment results and identify the applicability and efficiency of the different methods.

***Index terms*-- Passive one-way-delay measurements, IP metering, SLA validation**

I. INTRODUCTION

There exist a variety of motivations for performing passive measurements in IP networks. Many applications require the measurement of the Quality of Service (QoS) for the transport of specific IP flows or traffic aggregates. Network providers and customers are interested whether negotiated QoS values in SLAs are met (SLA validation). Measurements also provide the basis for usage-based accounting. Furthermore, measurement results are an important input for traffic engineering decisions.

SLAs are usually negotiated between network providers and customers or between neighboring network providers. One-way metrics like one-way delay are important for the observed quality especially for interactive applications. Therefore it is very likely that SLAs will contain guarantees for one-way metrics. Forward and return path for a data transmission may not have the same characteristics (even different paths can be used) and therefore can differ in quality. This means that one-way metrics cannot be derived directly from round-trip metrics, and that suitable measurement methods for one-way metrics are required for SLA validation. Since the traffic of interest is already present in the network, the measurement goal can be achieved efficiently by passive measurements avoiding the effort and the disadvantages of sending test traffic.

In this paper, we investigate requirements of a passive measurement system for one-way-delay measurements. Based on this we investigate how the selection of algorithms and parameters of the different building blocks influence the performance (resource consumption, speed) of the overall system. We focus on the analysis and comparison of different packet identifier (ID) generation functions. For this we investigate and compare the usage of (1) a simple combination of highly variable IP header bytes, (2) CRCs, (3) a simple hash function used in [DuGr00] and (4) the MD5 message-digest algorithm [RFC1321, Touc95]. We investigate these packet ID generation functions as part of a modular passive meter, which we have implemented based on an extended Linux NetFilter classifier [Russ00]. The meter is suitable for passive measurement of one-way-delay. It has a modular structure that allows us using and comparing different packet ID generation functions. We present the influence of these functions to the performance and the resource consumption of the system.

The paper is structured as follows. Section II compares the active and passive measurement approach and highlights the differences. Section III describes the requirements for passive one-way-delay measurements and shows the key building blocks of a passive measurement system. Section IV discusses how packet IDs can be generated most efficiently and gives an overview on related work. Section V describes our meter implementation that was used for the measurements presented in section VI. Section VII concludes the paper and identifies future work.

II. PASSIVE VS. ACTIVE MEASUREMENTS

Active measurements inject test traffic into the network in order to measure network characteristics. In contrast to this, passive measurements rely on the traffic that already exists in the network. In order to distinguish purely passive measurement methods from methods where packets are modified by the measurement process (e.g. by adding a timestamp to the packet) we call the latter semi-active measurement methods.

The development of active measurement methods to survey large parts of the Internet at high precision is an active field of research (e.g. [PaAM00], [KaZe99], [PaMA98], [UiKo97]). Active measurements give a prediction of the expected treatment of traffic in the observed part of the network. Active measurements are controllable experiments, which can be performed at any time and with any kind of traffic pattern that is of interest for the specific measurement objective [SeSK97].

Despite these advantages, active measurement methods have a number of disadvantages, mostly due to the approach of sending additional test traffic through the network under test. In order to obtain measurement results that are representative for certain applications, active measurements require the generation of appropriate (artificial) test traffic to emulate the expected traffic mix. In most cases this task is not trivial. Furthermore, test traffic always generates additional load on network links and routers and can significantly influence the measurement results. It may lead to additional costs if accounting is usage-based, and it might also irritate intermediate providers. Especially if test traffic is not recognizable as such, providers may suspect an attack. For SLA validation, care must be taken that injected test traffic is treated equal to regular traffic. Furthermore, if interdomain measurements are done, the test traffic should be generated in a way that a foreign provider cannot distinguish the test traffic from regular traffic. Otherwise providers could apply special treatment to the test traffic in order to influence the measurement results.

In contrast to this, passive measurements are based on the already existing traffic in the network. They provide a statement about the treatment of the current traffic in the observed network section. Since no test traffic is generated, passive measurements can only be applied in cases where the kind of traffic we are interested in is already present in the network. This is the case for most applications where statements about the actual situation in the network are required (like SLA validation, traffic engineering). Active measurements can always be applied supplementary, in order to predict the future network situation during times where no regular traffic is transmitted.

Passive measurements mainly have been used for simple tasks like packet counting (e.g. NeTraMet [RFC2123] [Brow97] and NetFlow [Cisc99]) and associated metrics, like volume, so far. The area of packet filtering and classification is an active field of research, which permanently comes up with new fast algorithms and methods to improve the filtering performance (e.g. [BoSS99], [GuMc99]). Passively measuring round-trip metrics is possible at a single measurement point by using packet pair matching techniques that rely on existing packet pairs like TCPdata/ACK, DNS request/response, etc. [Brow00]. Nevertheless, this method only works for protocols based on packet pairs. It requires classification based on higher layer information (which is for instance not possible if data is encrypted) and only measures round-trip metrics. Difficulties arise if two measurement points are involved in the measurement like for passive one-way delay (OWD) measurements.

III. PASSIVE ONE-WAY-DELAY MEASUREMENTS

An approach to realize passive OWD measurements is to generate a timestamp and a unique packet ID for each packet at two measurement points and to send this information to a control instance that calculates the delay. The packet ID is needed to associate the timestamps from the different measurement points to the correct packet.

For each packet that is measured a timestamp and a packet ID have to be generated, stored and transmitted to a collection point where the QoS computation takes place, based on the results from the different measurement points (Figure 1). Therefore the amount of measurement result data that arises per second depends on the number of measured packets n per second, the number of bits l_t used for the representation of the timestamp and the number of bits l_{id} for the packet ID.

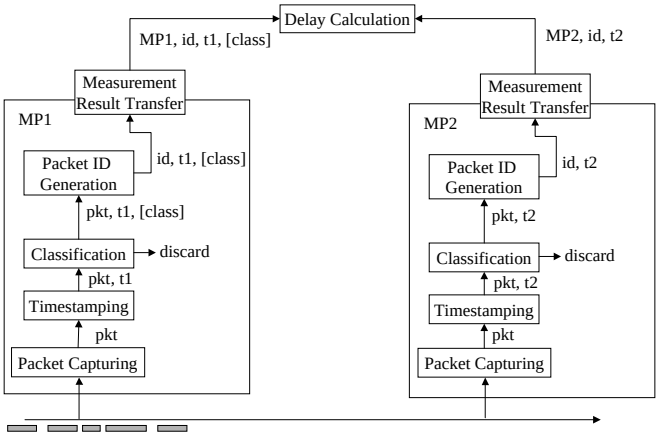

Figure 1: Passive One-Way-Delay Measurements

The main building blocks for implementing a one-way-delay measurement are shown in Figure 1. The processes involved are packet capturing, timestamping, classification, generation of a packet ID and transfer of measurement data. The requirements for the building blocks are examined in detail in the following subsections. Based on these building blocks, we investigate alternative methods for implementing them, in order to identify the most efficient way to perform the needed tasks.

Further issues that must be dealt with are privacy issues when capturing traffic from customers, difficulties in packet event correlation when packets are lost or duplicated, and the overall amount of measurement data captured and transmitted for evaluation.

A. *Packet Capturing*

A certain amount of bytes needs to be captured per packet as basis for the generation of a packet ID. The packet ID collision probability depends on the generation function and the number of bytes that are used as input. In [DuGr00] it is stated that the first 40 Bytes starting at the IP header are sufficient. However, the number of bytes that can be captured also depends on the processing power remaining for the measurement task. The packet capturing performance of a machine is limited by

- the number of interrupts generated by the NIC,
- the number of context switches,
- the amount of bytes transferred to user space,
- the current load of the machine (caused by other processes e.g. for the packet ID generation)

The number of interrupts depends on the number of packets that have to be passed from the hardware to the kernel. The number of context switches depends on the number of packets that have to be passed from kernel to user space. Both can be minimized through bundling or batching of packets.

In case of interrupt bundling, packets are queued in the NIC buffer until an interrupt is generated. Thus the accuracy decreases because timestamps are normally generated in the kernel and not on the NIC. The number of context switches can be reduced significantly by copying multiple packets at once (batching) instead of copying each packet separately from kernel to user space. The amount of bytes transferred to user space could be minimized through minimization of the number of bytes captured or by doing the packet ID generation in the kernel. Nevertheless in our experiments we observed that the processing performance is influenced only slightly by the number of bytes copied.

Since the packet ID generation function is called per packet, there is a tradeoff between the complexity of the function, the length of the packet ID and the remaining collision probability. Sampling can reduce the number of packets processed per time interval.

A dedicated metering device, which is attached to the network segment of interest, can use its full processing power for the measuring task. In case of a non-switched Ethernet subnet, the device can capture packets by operating its NIC in promiscuous mode, while in other cases an optical splitter or a monitoring port on a switch is needed for packet capturing. If measurements run as additional functions on a router, only a certain amount of processing capacity is available for measurements. It has to be taken care that the routing functionality is not negatively affected by measurements. Due to possible load changes it may be difficult to ensure that there will be always sufficient processing capacity for both routing and measurement tasks. Therefore we focus in the following on the first alternative.

In contrast to results in [ClDG00] we encountered no packet loss in our measurements on the hardware (Intel Fast Ethernet E100Pro) or the operating system kernel (Linux 2.4) as long as the test system is not in an overload state and the kernel socket buffers are sufficiently large to avoid buffer overflows during context switches.

B. *Timestamping*

A number of issues have to be considered for the basic function of assigning timestamps to packets for subsequent delay calculation. Internal buffering in the hardware and on the way through the kernel causes additional packet delay. Even if all involved measurement devices are equipped with the same hardware and operating system, packets can experience different delays (e.g. due to CPU load and the level of buffer filling). In order to reduce effects from additional variable delays, the timestamp should be assigned to the packet as early as possible.

A further problem that has to be solved when using two measurement points is clock synchronization between both points. Current solutions are based on the Network Time Protocol (NTP) [RFC1305], the Global Positioning System (GPS), and radio signals (e.g. DCF77). Each solution has its own drawbacks and advantages.

NTP is limited in its accuracy especially in large networks. Subsequent NTP packets may experience different queuing delays, resulting in relatively imprecise clock synchronization. GPS provides an adequate solution for the clock synchronization problem. It can be received everywhere on earth, and a GPS signal can achieve an accuracy with a maximum deviation of 100 nanoseconds. For the achievable accuracy one has to take into account that there is a loss in accuracy on the way from the GPS receiver to the clock tick in the kernel. Nevertheless, since delay values usually lie in the range of milliseconds, the achievable accuracy should be sufficient for the measurements. A certain drawback of GPS is that GPS receivers require a direct “intervisibility” with the satellites. This can become a severe problem since measurement devices are frequently positioned close to main nodes of the infrastructure, frequently located in server-rooms that may lack a window or may be located at the basement. Furthermore, GPS is still more expensive compared to other solutions.

A further possibility is to use radio signals for time synchronization. For instance the DCF77 radio signal, which is broadcasted at 77.5 kHz from a station near Frankfurt/Main in Germany, provides date and time with an accuracy of at least one millisecond. The signal can be received almost everywhere in Central Europe and is used amongst other purposes for the synchronization of public clocks (e.g. at railway stations). One problem with radio signals is that a proper reception is also not guaranteed everywhere. Furthermore, atmospherics may interfere with the signal. Due to its limitations in coverage, the DCF77 signal can only be used for measurements within Central Europe. But since radio signals all over the world are synchronized with each other, measurements with devices in different continents can be realized as long as any suitable local radio signal can be received in the area of interest.

A timestamp can be represented as absolute time. With this the number of bits needed for the representation of the timestamp depends only on the desired accuracy for the measured metric. A possibility to reduce the number of bits l_t used for the timestamp is to use relative timestamps. One

approach is to make an assumption on the maximum time t_{max} a packet needs to traverse the network from the ingress to the egress measurement point. With this upper limit the timestamp needs to be non-ambiguous only within this limit. In this case the value l_t depends not only on the desired accuracy of the time representation but also on the predetermined limit for the maximum time the packet needs to traverse the network. Another possibility is to use an absolute timestamp only for the first packet in a given time interval $[0,t_{int}]$ and use timestamps relative to this for successive packets that arrive in the same interval.
Further issues are that the time that is needed for the timestamping process can differ for subsequent packets, which can also lead to inaccuracy [ClDG00].

C. Classification

A classification of packets is required if only selected packets are used for the measurement. A pre-selection is useful to reduce the amount of resulting measurement data and the required processing time for the subsequent processes (like packet ID generation). It also could be considered to place the classification before the timestamping in order to relieve the timestamping process. But this would contradict the requirement to do the timestamping as early as possible.
Classification can filter out packets with specific characteristics. This can be for example all packets that belong to a specific flow (characterized by a common quintuple [src,dst,src_port,dst_port,proto]) or traffic aggregate (characterized by a common DiffServ Codepoint) to determine the QoS for specific applications or traffic classes. Another possibility is to select packets according to the packet characteristics (e.g. packet length) or to bit patterns within the packet to achieve sampling [DuGr00].
In certain cases it is important to maintain the information to which flow or class the measured packet belongs to. For instance if the quality for different DiffServ traffic aggregates is measured simultaneously it is desirable to keep information about the class together with the packet ID and timestamp. If the packet ID is generated with a bijective function on the header (e.g. compression), the additional information can be extracted from the packet ID directly.
In all other cases, the needed information has to be transferred to the analysis application in addition to the packet ID. Since the packet ID provides a unique mapping to the packet, this additional information does only need to be transferred from one measurement point. The other measurement point can determine the information via mapping its packet ID to the information stored under the same packet ID at the analysis point.

D. Packet ID Generation

Unlike semi-active measurements, passive measurements are based on methods where packets are neither marked nor modified in another way. Therefore the recognition has to be based on fields that already exist in the packet. In order to get the same packet ID for one packet at both measurement points the packet ID generation should be based only on fields that are invariant or predictable during the transport. Fields that are highly variable between the packets (e.g. the datagram ID) are more suitable than fields that are nearly constant or vary only between a few values (e.g. version field). The generation of a packet ID should be based on fields that

- already exist in the packet (no modification of the packet),
- are invariant or predictable during the transport (at least on the path from the ingress to the egress second measurement point) and
- are highly variable between the different packets.

The request for low collisions (uniqueness of the ID) contradicts the request for a small packet ID, because the more bits are used for representing the packet ID the lower is the probability of collisions. The collision probability within a traffic trace depends on

- the distribution of the bit sequences taken as input to the packet ID generation (that means it is highly dependent on the considered traffic mix),
- the packet ID generation function,
- the size of the packet ID l_{id},

the used Operating System (OS) (if the datagram ID is considered, see section IV.C)
The goal is to achieve an acceptable low probability of collisions with a packet ID that does not exceed the available capacity for the measurement result data transfer. As for the timestamp, the packet ID only needs to be unique in the given time interval $[0, t_{max}]$. This limits the number of possible combinations to the number of packets n_{max} that can be observed within thid interval. For example for a 155 Mbits/s link with an average packet size of 512 Bytes and a maximum time to traverse the network of 10s, n_{max} would be 378,420 packets. This amount of combinations can be represented by 19 bits.
In most cases we can make some assumptions about the bit sequences (unprocessed packet fields) that we want to transfer into packet IDs:

- IP header fields and certain combinations of them are often limited in their variance on one link (e.g. what addresses occur).
- For some fields certain values have a very low probability to occur (e.g. certain addresses or combination of flags).
- IP header fields (and also the transport header) for packets of one data flow are often very similar (e.g. source, destination, protocol, port stay the same [RFC2507]). That means that the headers of (more or less) successive packets are often similar, especially if only a few flows are active.

With these assumptions about the variance of the original bit sequence the number of bits for the packet ID can be limited according to the possible occurring combinations. Knowledge about the expected traffic mix therefore can reduce the number of required bits. Furthermore, the maximum number of bits $l_{id,\ max}$ that can be allocated for a packet ID depends on

- the rate P_S at which the selected packets occur on the link,
- the available bandwidth for the measurement result transfer L_M,
- and the number of bits l_t needed to represent the timestamp.

The overall packet rate P_T depends on the packet size s and the rate for data traffic L_D on the link. The maximum packet rate $P_{T,max}$ can be calculated from the maximum data rate and the smallest packet size s_{min}:

$$P_{T,\max} = \frac{L_D}{s_{\min}} \quad (1)$$

The packets of interest are the packets that were selected for the measurement. This can be all packets on the link, packets selected according to certain characteristics (e.g. all packets that belong to a specific flow, all packets with a specific size, etc.) or packets selected in accordance to random patterns. The average packet rate for the selected packets (P_s) can differ significantly from the average packet rate for all packets on the link (P_T). The selection of only specific packets can lead to a less frequent occurrence of bursts. In that case smaller buffers (e.g. for buffering timestamp and packet ID for each packet) may be sufficient to handle the capturing of selected packets.

Nevertheless, the maximum possible packet rate for the selected packets $P_{S,max}$ equals the maximum packet rate for all packets on the link, $P_{T,max}$ because it can usually happen that two packets that meet the selection criteria directly follow each other. Whether this happens is of course not only dependent on the traffic mix but also on the selection strategy. For instance a selection pattern that filters out only packets of one flow with TCP sequence numbers that are a multiple of 50, would probably ensure a certain minimum distance between two selected packets. Nevertheless it has to be taken into account that packets can be re-ordered or lost when estimating the minimum distance between selected packets.

The minimum content of a measurement report is the timestamp and a packet ID per packet. This leads to a minimum size of $l_{total}=l_{id}+l_t$ bits per packet in the measurement report. We make the following assumptions in order to calculate the maximum number $l_{id,max}$ of bits a packet ID can have:

- The maximum packet rate of the selected packets is equal to the maximum packet rate on the link ($P_{S,max}= P_{T,max}$).
- The bandwidth available (or reserved) for the measurement result transfer L_M is only a fraction r of the overall bandwidth of the link L_D

$$r = \frac{L_M}{L_D} \quad (2)$$

- Measurement reports are transmitted immediately after generation (no buffering).
- Bursts of packets with smallest size s_{min} can occur.

If one report is generated per packet, the maximum report rate $P_{R,max}$ equals the maximum packet rate $P_{T,max}$. On the other Hand $P_{R,max}$ should not exceed the available rate for the report transfer L_M/l_{total}:

$$P_{R,\max} = P_{T,\max} = \frac{L_D}{s_{\min}} \quad (3)$$

$$P_{R,\max} = \frac{L_M}{l_{total}} = \frac{L_M}{l_{id} + l_t} \quad (4)$$

$$\frac{L_D}{s_{\min}} = \frac{L_M}{l_{total}} = \frac{L_M}{l_{id} + l_t} \quad (5)$$

With this the maximum number of bits allowed per packet ID is given by:

$$l_{id,\max} = (r \cdot s_{\min}) - l_t \quad (6)$$

So we are looking for a recognizable unique ID that consists of a maximum number of $l_{id,max}$ bits.

E. Measurement Result Transfer

In order to calculate QoS parameters like delay, two timestamps have to be compared. If more than one measurement point is involved the measurement results (timestamps and packet ID) from the different measurement points have to be collected at a common location in order to calculate the delay value. This collection point can be located on a separate host. It also can be co-located with one of the meters in order to reduce the amount of data that has to be transferred. The following possibilities to transfer the measurement results have to be distinguished:

a) In-packet: The measurement results (timestamps and packet ID) from the first measurement point for packet recognition and the timestamp are carried within the packet.
b) In-band: The measurement results are sent directly on the same path as the data.
c) Out-of-band: The measurement results are sent on a separate path.

Solution a) requires the modification of packets. This would lead to a semi-active measurement method with all its disadvantages (e.g. the need for packet modification at link rate). The advantage is that the analysis can take place directly at the second measurement point. This method is efficient especially if packets already contain information that can be used for the packet event correlation (e.g. an IPsec authentication header).

Solution b) leads to additional load on the network under test. That means measurement result data packets can influence the original data flow and can lead to the same disadvantages that active measurements observe. The in-band sending of packets with measurement results therefore somehow contradicts to the passive approach where no influence on the original traffic is desired. It could be seen as a special form of semi-active measurements, where

measurement data is send in separate packets instead of including the information directly into the data packets (by modifying them). Nevertheless, there is a difference between sending test traffic for active measurements and the transmission of measurement results. The amount, type and timeframe for the sending of test traffic for active measurements is dictated by the measurement task. In contrast to this, the sending of measurement results can be controlled by other means. For instance it could be sent with a lower priority (e.g. lower-than-best-effort class) or only at times when the network is lightly loaded, or routed over paths that are currently not loaded (e.g. via MPLS). Which alternative is to be preferred depends on the policy for the evaluation of the metric (e.g. real-time or non-real-time).

Solution c) requires a separate path to the analysis application. This can be achieved for instance via a second interface card and a separate measurement network that connects all measurement points. This approach does not influence the data traffic but requires capacity in a different network.

In all three cases additional capacity (either on the existing network or on a separate network is required). For economic reasons even a separate reporting network would probably have a lower capacity then the "production network". Therefore to save resources (storage capacity and bandwidth) the measurement result traffic should be kept as low as possible.

IV. PACKET ID GENERATION

A. Requirements

In order to reduce the number of bits used for the packet ID an appropriate function can be used to transform the considered Bytes of the packet into a smaller ID. The transformation should be done in a way that

- the resulting ID is as small as possible,
- the probability for collisions is as low as possible and
- the ID generation is fast.

Furthermore, it would be advantageous (but not required)

- to use an operation that always leads to an ID with the same fixed length. This would ease the handling, transmission and the estimation of the overhead caused by measurement result transport.
- to use a bijective operation. With this the original bit sequence could be derived from the ID. Certain attributes like source and destination could be stored without transferring them in addition to the ID. Nevertheless a bijective function would probably require too many bits for the packet ID and can only be used if substantial assumptions can be made about the traffic mix in advance.

B. Considered Packet Fields

In the passive approach measurement packets are not modified. That means that only already existing fields of the packets can be used to compute a packet identifier. For the generation of a packet ID different fields of the packet and different methods can be used. In order to generate a unique packet ID that can be recognized at the second measurement point only the parts in the IP packet that do not change on the way to the receiver (immutable during transport) can be taken into account. Fields that are mutable but predictable could also be used for the packet ID generation. Furthermore, it is advantageous to consider fields that are highly variable between different (successive) packets. Whether fields with low variability (e.g. version field) should be considered in the ID calculation depends on the implementation (i.e. they should be included as long as there is no significant performance decrease).

The following table shows the IP header fields, their immutability on the path and their variability between different packets. The last column indicates whether we consider this field in our packet ID generation functions (see section VI).

Header Field	Immutability on the Path	Variability Between Packets	Considered
Version	Yes	Extremely small (IPv4, IPv6)	No
Header Length	Yes	Small (only if options are present)	No
Type of Service (TOS)	No (some routers change this field)	Can be high (but usually not used)	No
Total Length	Yes	Can be high (although certain packet sizes have a higher probability than others e.g. 40, 552, 576, 1500 bytes [nlanr97])	Yes
Datagram ID	Yes	High	Yes
Flags	No (intermediate routers may set the "don't fragment" flag)	Moderate	No
Fragment Offset	No (changed if re-fragmentation is done => rare but possible)	Can be high (depends on amount of fragmentation and packet size distribution)	No
Time to Live (TTL)	No (decrements at each router)	Can be high	No
Protocol	Yes	Small	Yes
Header Checksum	No (changes always if other header-fields changed)	Can be high	No
Source Address	Yes	Can be high	Yes
Destination Address	Yes	Can be high	Yes

Table 1: Immutability and variability of IP header fields

The TTL, the header checksum and also the Type-of-Service byte are mutable on the path [RFC2402] and therefore excluded from the packet identification. The flags field and the fragment offset are also mutable due to fragmentation and because routers are allowed to set the "don't fragment" bit. Since fragmentation splits up one datagram into multiple fragments, it makes the generation of a unique packet ID more complicated and let the question arise whether datagrams or fragments should be considered for the measurements (see section IV.C). In [DuGr00] it is

assumed that fragmentation is done at the network edges only. Therefore in their approach flags and fragment offset are included in the packet ID generation. Furthermore, the IP header may contain options. Some of these are mutable and therefore should not be included in a packet ID generation [RFC2402]. In [DuGr00] the presence of variable options is neglected since they are used only rarely.

C. Datagram ID and Fragmentation Issues

The datagram ID usually is the most variable field of the IP packet header. Therefore it is very well suited as one base field for the packet ID generation. Nevertheless there are certain issues that have to be considered when using the datagram ID for the packet ID generation. The datagram ID is represented as a 16 bit field. This leads to a wraparound at least after 65535 packets from the same source to the same destination (dependent on the implementation whether a global or per destination counter is used).

The datagram ID is only unique for a specific combination of source, destination and protocol fields (in some cases only the source address or source and destination are used). If packets from different sources or to different destinations are measured, source and destination should also be considered in the packet ID generation.

If fragmentation takes place on the way from the sender to the receiver, fragments with the same ID will occur on the network. In this case the fragment offset could be included in the packet ID generation to reduce the collision probability. A more problematic case occurs if packets are fragmented or re-fragmented on the way between the ingress to the egress measurement point. A solution is to instruct both meters to always re-assemble fragmented packets. If re-fragmentation can be excluded it would be sufficient to instruct only the egress to re-assemble the packets.

Simply excluding fragmented packets from the measurement is not a solution. For SLA validation we are looking for maximum delay values to compare them with thresholds. Therefore especially packets for which high delay values are expected should be considered for the measurement. Since fragmentation takes time it is likely that especially fragmented packets will experience higher delays. Furthermore, large packets experience higher delay values anyway [RFC889] and at least the first fragment of a packet has the size of the MTU on the link.

The datagram ID for retransmitted TCP packets can be equal to the one in the original packet [RFC791]. Since the packets are indeed equal [DuGr00] there is no way to distinguish them. In this case the arrival times at the measurement points (represented in the timestamp) might be used to decide which of the equal packet IDs on the different measurement points could belong to the same packet. Nevertheless in this special case it is impossible to ensure that a correct packet correlation is performed.

The assignment of datagram IDs depends on the operating system. In Solaris, FreeBSD and Linux up to Kernel 2.2 the datagram ID is increased per source/destination pair or in old versions a global counter has been used. Due to security reasons this behavior was changed in some operating systems. Predictable IDs are problematic because they can be used to discover the number of hosts behind a firewall/NAT and simplify certain attacks (e.g. DNS poisoning). For this reason OpenBSD implements a method that leads to more random values in the datagram ID instead of increasing the ID by one per packet. Linux 2.4 simply sets the “don’t fragment” flag for packets that are smaller or equal to the path MTU (PMTU) and therefore don’t need to be fragmented. The datagram ID for these packets is set to zero. In case the PMTU cached by the Linux kernel per source destination pair changes the source gets an ICMP message, the PMTU is updated and the packet is fragmented at the source and retransmitted. Nevertheless [RFC791] states that an identifier has to be chosen that is “unique for this source destination pair and protocol for the time the datagram (or any fragment of it) could be active in the network". Although Linux is still perfectly interoperable it does not comply to this statement. In addition it might be difficult to fulfill this requirement anyway in future high speed networks providing Gigabit speed with only 16 bit for the identifier. With a datagram ID set to zero the collision probability increases for packet IDs that include the datagram ID. Since the majority of the Internet traffic is not fragmented due to the PMTU discovery the datagram ID is not usable for packet identification on new Linux kernels. The different implementations of the operating systems lead to a collision probability (for packet IDs based on the datagram ID) that is not only dependent on the traffic mix, but also on the used operating system and kernel version.

D. Related Work

In [GrDM98] specialized ATM hardware is used for passive one-way delay measurements. Packet recognition is based on a 32-bit AAL5 CRC calculated in hardware over the ATM-cell payload. The amount of duplicate cells was found to be less than 1 percent in traces with several million cells from NFS traffic.

In [DuGr00] a hash function over the invariant header fields and parts of the payload is used to generate a packet ID. It is shown that with their traces the consideration of the first 40 Bytes is sufficient to reduce the probability of duplicate packet IDs to less than 10^{-3}.

E. Packet ID Functions

There are different possibilities to generate a packet ID. Section IV.B already addresses the issue which parts of the packet should be chosen for an ID generation. This section deals with the ID generation function itself. Therefore in this section the functions are compared by assuming that always the same selection of packet fields is used as basis for the calculation of the packet ID. There are different possibilities to generate a packet ID:

- Unprocessed selected header fields
- One-way hash functions (DuGr00, MD5, SHA-1, etc.)
- Checksums, CRC
- Compression functions

Functions that map a large bit sequence (part of the packet) to a smaller bit sequence (the packet ID) can always increase the collision probability. Especially if no assumptions can be made about the original bit sequence it is difficult to avoid collisions. Using the selected fields of the packet without further processing means to use the calculation basis itself instead of a derived ID. This would lead to the minimum collision probability that ever can be achieved with the given traffic mix. Furthermore, it would reduce the required processing power because no function has to be performed on the selected fields. Nevertheless this method would result in a packet ID size l_{id} that is equal to the sum of the bits of the selected fields. Especially for small packets this would increase the rate required for the measurement report messages up to the rate for the data flow itself (see section III.E for discussion on this).

The hash-function used in [DuGr00] is a simple modulo operation over the first 40 Bytes of the IP packet.

The message digest MD5 [RFC1321] is a cryptographic hash function that generates a 128-bit fingerprint from a message with an arbitrary length. It is mainly used for message authentication. Such hash functions are often used together with digital signatures in order to make these more resistant against forgery. Cryptographic hash functions are designed to be collision resistant in a way that it is nearly impossible to find two messages that have the same message digest. In [RFC1321] the difficulty of finding two messages that have the same message digest is denoted to be on the order of 2^{64} operations. The difficulty in finding a message that has a given message digest is even higher (on the order of 2^{128} operations). It is important to notice that collision resistance here just means that it is nearly impossible to generate a message that would result in a specific digest (which simply means one cannot find a reverse function). The fact that it is difficult to find two messages that lead to the same digest does not imply that it would be impossible that by accident two messages would result in the same digest, leading to a collision. However MD5 provides a good distribution of hash values, which makes collisions rare.

An advantage for using the MD5 hash function for the packet ID generation is the re-usability for IPsec. MD5 is one alternative to calculate the Integrity Check Value (ICV) in the IPsec Authentication Header [RFC2402]. If MD5 is suitable as packet ID, a semi-active one-way-delay measurement can be realized by using the ICV as ID for the packet event correlation. Furthermore, the AH contains a 16 bit field reserved for future use. This field can be used to carry a timestamp from the first to the second measurement point. If the delay calculation takes place at the second measurement point, no further measurement result data transfer is required.

CRCs have been developed for error detection and correction. That means if messages differ in only a few bits, the CRC of these messages should differ in order to indicate errors. Nevertheless CRCs were not designed to be collision resistant. In contrast to cryptographic hash functions it is possible to generate two files that would produce the same CRC. The frequency of collisions depends on the size of the CRC and the generator polynom.

If substantial assumptions can be made about the expected traffic, the selected packet fields can be converted into a packet ID by using a compression function. This procedure would maintain the lowest possible collision probability that can be gained with the unprocessed fields. The applicability of a compression function always implies redundancy in the original message. Indeed redundancy can be found if assumptions about the expected traffic can be made in advance. Fist of all, one can apply general IP rules to exclude specific bit combinations. For instance the source address field should not contain a multicast address, and private IP addresses should not occur in a backbone network. Furthermore, one can make assumptions due to reality and experience (e.g. version is either 4 or 6, TOS field not used, TTL maximum value etc.). The next step would be to make assumptions about the expected traffic mix (e.g. expected combinations of source and destination addresses). The degree of achievable compression determines the required size of the packet ID. It heavily depends on the number and scope of the assumptions. An advantage of (lossless) compression functions is that they are bijective. This means if one needs further information on the measurement (e.g. destination addresses) one can extract this information directly from the packet ID. Furthermore, successive packets of a data flow contain time redundancy. For instance for one IP flow the quintuple source address, destination address, protocol and the port numbers remain the same in all packets. This can be utilized if measurements are done in a flow specific way (e.g. one-way delay per flow is needed).

V. METER IMPLEMENTATION

The design of a flexible IP meter has to take several requirements into consideration:

- Classification speed (time taken per packet),
- Classification functionality (with respect to filter attributes),
- Packet evaluation functionality (with respect to calculation of flow properties),
- Ease of extensibility (for adding new functionality) and
- Control and data access interfaces of the meter system.

For the design of our meter system we put the focus on flexibility and extensibility while retaining acceptable classification speed. Figure 2 shows the core of the meter implementation without the external interfaces.

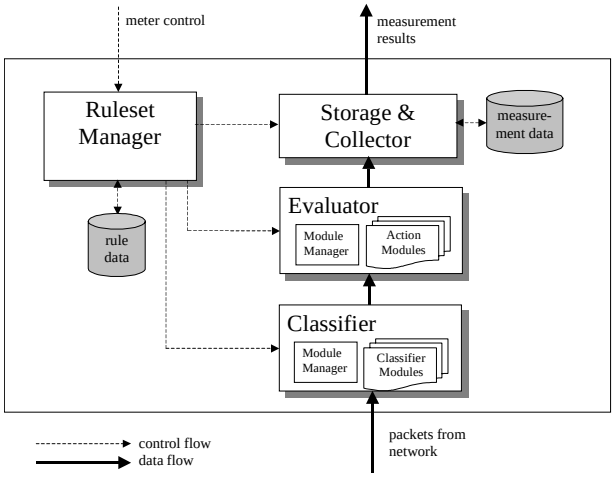

Figure 2: Meter Implementation

The Ruleset Manager processes and stores incoming filter rule descriptions. The description has to be syntax-checked and processed to an internal format before storage in the meter's rule database. The Ruleset Manager instructs the other core components according to the given metering task. The Classifier's task is to manage the set of installed filter rule patterns and to identify the matching rule(s) for incoming packets. The classifier component delivers the packet data and corresponding metadata (length, timestamp, rule ID) to the Evaluator for further analysis (i.e. calculation of flow properties). The Evaluator component is responsible for the execution of the actions specified in a complete filter rule description and is composed of a Module Manager and Action Modules. The filter action tells which metrics shall be calculated for that specific flow. For each metric (e.g. volume, bandwidth, jitter, RTT) an Action Module implements the required algorithm. The Module Manager has to identify which Action Modules have to be used and to apply those actions onto the corresponding flow data. Furthermore it has to make sure the Action Modules required for the installed filters are present and properly initialized. The Collector will be instructed by the Ruleset Manager to query the accumulated measurement data (e.g. volume counters) from the flow database inside the Evaluator. Data collection tasks (e.g. retrieve accounted volume every 60 seconds for flow A) are scheduled in the Collector so that data is retrieved at the correct point in time specified by the filter rules. The Collector forwards the measurement results to the Storage component. The Storage component gathers the measurement data sent to it by the Collector and stores them into an internal repository. The repository inside the Storage allows to accumulate measurement data from a number of measurement intervals (e.g. 20 volume counters taken once every minute). Accumulation of measurement data allows to reduce the number of result packets transmitted.

We decided to base our first classifier on the existing NetFilter classifier [Russ00], which is running entirely in the Linux 2.4. kernel. Different classifier modules allow matching of additional packet fields (e.g. RTP header info) apart from the standard attributes (SrcIP, DstIP, SrcPort, DstPort, Protocol). The NetFilter implementation has been enhanced to support measurement on a dedicated device (via promiscuous mode) and to minimize context switches. Additionally we have developed new classifier modules (e.g. for matching RTP packets). Using NetFilter as classifier limits the applicability of the meter to the Linux operating system. Since the classifier interface is well defined it would be possible to use libpcap [PCap00] and a portable classifier implementation to run the meter on other UNIX platforms (e.g. FreeBSD, Solaris).

Having the classifier in the kernel means timestamping and metadata generation is done in the kernel. Therefore we get the most accurate timestamps (as possible in software) and access to all important metadata (e.g. incoming/outgoing interface etc.). However the main advantage of classification in the kernel is that only the interesting packets (i.e. packets that are matched by a classification rule) are passed to user space. Therefore we have no unnecessary context switches. The number of context switches is further decreased due to batch delivery. The snap size (i.e. the number of bytes of each packet that is passed to user space) can be configured so that data copying can be reduced to the necessary minimum.

We have decided to do packet evaluation in user space for a number of reasons. First evaluation might get quite complex (e.g. packet ID generation) and we want avoid bloating the kernel with complex functions. Putting complex functions in the kernel might also decrease performance due to blocking problems. Since there are numerous evaluations possible it should be easy to add a new action module to the meter. Writing such an extension as kernel module is far more complex than implementing a shared library. However, doing evaluation in the kernel could save some more processing time depending on the task. In the case of packet ID generation we could for example further reduce context switches and copy operations because we would then only need to copy the packet ID to user space instead of the first part of the packet used as input for the ID generation function. However, we think that the performance gain is only marginal compared to the disadvantages mentioned above.

Our meter implementation has building blocks with well-defined interfaces so that single parts of the meter can be exchanged. The building blocks are realized as C++ classes. Furthermore, our meter is very flexible because we use loadable modules for the classification and evaluation. In fact these are shared libraries and Linux kernel modules that could be added or even exchanged during runtime. The interface for these modules have been well defined so that new modules can be written without knowing the internals of the meter.

VI. MEASUREMENTS

This section presents the measurements we did with the meter implementation described in section V. We have implemented the following packet ID generation functions: Unprocessed fields, a 16 bit hash function, CRC-16, CRC-

32 (from AAL5), the standard 128 bit MD5 and a folded 32 bit MD5.

The packet ID is generated on a basis of 40 Bytes that consist of the header fields selected in section IV.B (total length, datagram ID, protocol, source and destination address) and 27 Bytes of the payload. We use an absolute timestamp represented by 64 bits (struct timeval). Approaches to reduce the number of bits by using relative timestamps are dependent on the considered network and are currently not considered in our implementation. In order to compare the possible packet ID generation functions we use the following comparison criteria:

- Size of the resulting ID
- Probability of collisions
- Processing time to generate ID
- Effort to encode additional info (like source and destination address) in the ID

A. *Test Setup*

For the test setup we use a SUN Sparc 20 machine for traffic generation and a Linux host as destination (Fast Ethernet). Another Linux host (PIII-550, Intel Fast Ethernet E100Pro, Kernel: 2.4), which is connected to the same hub, is used as measurement device. The traffic flow generation is based on an ATM test system developed at GMD FOKUS. The test system includes a special hardware, the TANYA ATM test interface [KrMT99, CaTZ00]. Our meter implementation described in section V is running on the measurement device. The architecture of the testbed with the meter and flow generator is shown in Figure 3.

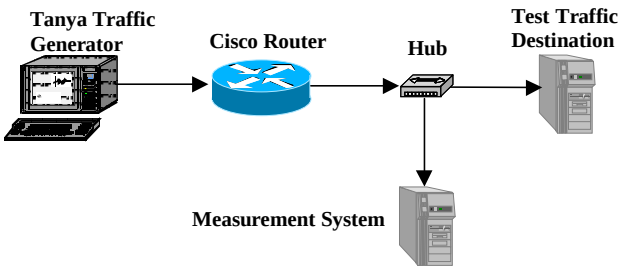

Figure 3: Test Setup

For the first test we use a synthetic flow with 120,000 UDP packets with a size of 64 bytes each generated by the TANYA traffic generator. The different packet ID generation modules are loaded and for each packet seen by the meter a timestamp and a packet ID is generated. For a second test we used an RTP video flow, which consists of 83,114 packets.

B. *Packet ID Generation Performance*

For measuring the Packet ID generation performance we measured the time that is needed for each packet to generate the ID. The average packet ID generation time for each function is shown in Table 2.

C. *Packet ID Collision Probability*

For very first tests on the collision probability of the different packet ID generation functions we calculated the number of collisions that occurred in the measured traces. The results are shown in Table 2. Since the collision probability highly depends on the traffic mix we plan to perform further tests with traffic traces captured in different networks (see section VII).

D. *Results*

Table 2 shows the results of the comparison of the different packet ID generation functions.

	Unprocessed Fields	**CRC-16**	**16-bit Hash**
Size of ID	320 bit	16 bit	16 bit
Collision Probability (artifical traffic)	0.0	0.835283	0.991543
Average processing time (artifical traffic)	0s (used as reference)	2.84855 us	1.76284us
Collision Probability (RTP Flow)	0.0	0.716799	0.992769
Average processing time (RTP Flow)	0s (used as reference)	2.89509 us	1.74516 us
Effort to store source, dest	0	64 bit	64 bit
	CRC-32	**Folded MD5**	**MD5-128**
Size of ID	32 bit	32 bit	128 bit
Collision Probability (artifical traffic)	0.0	0.0	0.0
Average processing time (artificial traffic)	1.86471 us	5.46122us	5.75711us
Collision Probability (RTP Flow)	0.000024	0.000024	0.0
Average processing time (RTP Flow)	1.85348 us	5.46099 us	5.56622 us
Effort to store source, dest	64 bit	64 bit	64 bit

Table 2: Comparison of Different Packet ID Generation Functions

If the plain fields are used, no additional processing time is required, but the ID would have to consist of 320 bit. On the other hand if a small ID of 16 bit (CRC or simple hash) is used only 2^{16} identifiers can be represented. As expected many collisions can be observed in this case, because the used traffic traces contain much more packets than IDs possible. If the ID is extended to 32 bit the collisions are reduced to zero for the used artificial traffic. For the RTP flow one collision occurred with the CRC-32 and the folded MD5. As expected the average ID calculation time remains nearly the same in both experiments. The MD 5 operations take about 3 times as long as the CRC calculation. The folding operation to generate a folded MD5 seems to be negligible small.

VII. Conclusions and Future Work

In this paper we evaluated the building blocks needed for passive one-way-delay measurements. We investigated what parameters influence the packet capturing, the timestamping and the classification process. We then identify the requirements for the selection of suitable functions for the generation of a packet ID, which is used for the packet event correlation at the involved measurement points. On one hand the ID has to be sufficiently small to not exceed the available capacity for the measurement result data transfer. On the other hand the probability of collisions (occurrence of duplicate IDs) should be minimized for the given scenario.

We first investigate which packet header fields are most suitable as basis for an ID generation function. Then we compare different functions for generating a packet ID with the approach of using the unprocessed packet fields as identifier. We have developed a flexible meter that allows us to plug in different packet ID generation functions as independent modules. We implemented a simple 16-bit hash function, a CRC-16, a CRC-32, the standard 128-bit MD5 and a 32-bit folded MD5 function as modules. First measurements show the differences in terms of processing time and collision probability for the different functions.

We plan to continue this work by evaluating the functions against large traffic traces captured in different networks and with varying number of bytes as basis for the packet ID generation. The meter will be extended with control functions to dynamically configure the appropriate functions that have to be loaded for a given measurement task. Furthermore, we are working on the implementation of semi-active measurement methods and will compare them with the purely passive approach.

VIII. Acknowledgements

We would like to thank Carsten Schmoll who designed and implemented the largest part of the meter we used for the measurements and Lutz Mark for helping us with the test traffic generation.

IX. References

[BoSS99] Niklas Borg, Emil Svanberg, Olov Schelen: Efficient Multi-field Packet Classification for QoS Purposes, IEEE/IFIP Seventh International Workshop on Quality of Service IWQoS '99 UCL, London, Jun 1 - June 4, 1999

[Brow00] Nevil Brownlee: Packet Matching for NeTraMet Distributions, Presentation at RTFM Get-Together, IETF Adelaide, March 2000, http://www.auckland.ac.nz/net/Internet/rtfm/meetings/

[Brow97] N. Brownlee: Reference Manual NeTraMet & NeMaC Version 4.1, Information Technology Systems & Services, The University of Auckland, New Zealand, November 1997 (http://www.auckland.ac.nz/net/Accounting/ntm.Release.note.html)

[CaTZ00] Georg Carle, Jens Tiemann and Tanja Zseby: Assessment of Accounting Meters with Dynamic Traffic Generation based on Classification Rules, in proceedings of The First Passive and Active Measurement Workshop (PAM2000), pages 127-133. The University of Waikato, New Zealand, April 2000.

[Cisc99] NetFlow Services and Applications, White Paper, Cisco Systems, 1999

[ClDG00] John Cleary, Stephen Donnelly, Ian Graham, Anthony McGregor, Murray Pearson: Design Principles for Accurate passive Measurement, The First Passive and Active Measurement Workshop PAM 2000, Hamilton, New Zealand, April 3-4, 2000

[DuGr00] Nick Duffield, Matthias Grossglauser: Trajectory Sampling for Direct Traffic Observation, Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden, August 28 - September 1, 2000.

[GrDM98] Ian D. GRAHAM, Stephen F. DONNELLY, Stele MARTIN, Jed MARTENS, John G. CLEARY: Nonintrusive and Accurate Measurement of Unidirectional Delay and Delay Variation on the Internet, INET'98, Geneva, Switzerland, 21-24 July, 1998

[GuMc99] Pankaj Gupta and Nick McKeown: Packet classification on multiple fields, Proceedings of ACM SIGCOMM'99, ACM, August 1999.

[KaZe99] S. Kalidindi, M. Zekauskas: Surveyor: An Infrastructure for Internet Performance Measurements, Proceedings of INET'99, San Jose, CA, USA, June 22-25, 1999

[KrMT99] N. Kroth, L. Mark, J. Tiemann: A Framework for Testing IP QoS over ATM Networks: Implementation and Practical Experiences, Proceedings of the 2nd International Conference on ATM (ICATM'99), Colmar, France, 21 - 23 June 1999

[nlanr97] http://www.nlanr.net/NA/Learn/packetsizes.html

[PaAM00] V. Paxon, A. Adams, M. Mathis: Experiences with NIMI, The First Passive and Active Measurement Workshop (PM 2000), Hamilton, New Zealand, April 3-4, 2000.

[PaMA98] V. Paxon, J. Mahdavi, A. Adams, M. Mathis: An Architecture for Large-Scale Internet Measurement, IEEE Communications Vol 36 No 8, p. 48-54, August 1998

[PCap00] tcpdump/libpcap, http://www.tcpdump.org

[RFC1305] D. Mills: Network Time Protocol (Version 3) specification, implementation and analysis, RFC 1305, University of Delaware, March 1992.

[RFC1321] R. Rivest: The MD5 Message-Digest Algorithm, RFC1321, April 1992

[RFC2402] S. Kent, R. Atkinson: IP Authentication Header, RFC 2402, November 1998

[RFC2507] M. Degermark, B. Nordgren, S. Pink: IP Header Compression", RFC 2507, February 1999

[RFC791] Jon Postel (Editor): Internet Protocol, RFC 791, September 1981

[RFC889] D. Mills: Internet Delay Experiments, RFC 889, December 1983

[Russ00] Paul Russel: Linux 2.4 Packet Filtering HOWTO, http://netfilter.samba.org/ unreliable-guides/packet-filtering-HOWTO.html, May 2000

[Touc95] Joseph D. Touch, Performance Analysis for MD5, Proceedings of ACM SIGCOMM'95, Cambridge, Massachusetts, USA, August 28 - September 1, 1995

[UiKo97] Henk Uijterwaal, Olaf Kolkman: Internet Delay Measurements using Test Traffic - Design Note, RIPE NCC, Document RIPE-158, May 1997