Application Level Measurement

Patrik Arlos

Blekinge Institute of Technology,
37179 Karlskrona, Sweden
http://www.bth.se

Abstract. In some cases, application-level measurements can be the only way for an application to get an understanding about the performance offered by the underlying network(s). It can also be that an application-level measurement is the only practical solution to verify the availability of a particular service. Hence, as more and more applications perform measurements of various networks; be that fixed or mobile, it is crucial to understand the context in which the application level measurements operate their capabilities and limitations. To this end in this paper we discuss some of the fundamentals of computer network performance measurements and in particular the key aspects to consider when using application level measurements to estimate network performance properties.

Keywords: Application level measurements, Computer network measurements, Network performance measurments, Accuracy, Quality.

1 Introduction

In recent years computer network measurements (CNM), and in particular application level measurements (ALM), have gained much interest, one reason is the growth, complexity and diversity of network based services. CNM/ALM provide network operations, development and research with information regarding network behaviour. The accuracy and reliability of this information directly affects the quality of these activities, and thus the perception of the network and its services [1],[2].

Measurements are a way of observing events and objects to obtain knowledge. A measurement consists of an observation and a comparison. The observation can be done either by humans or machines. The observation is then compared to a reference. There are two types of references; personal and non-personal. A personal reference is formed by the individual based on his experiences. A non-personal reference has a definition that is known and used by more than one individual, for instance the International System of Units (SI) [7] provides a set of global references.

2 Network Performance Framework

The Network performance framework consists of four modules; generation, measurement, analysis and visualization of analysis results. The framework is depicted

D. Kouvatsos (Ed.): Next Generation Internet, LNCS 5233, pp. 14–36, 2011.

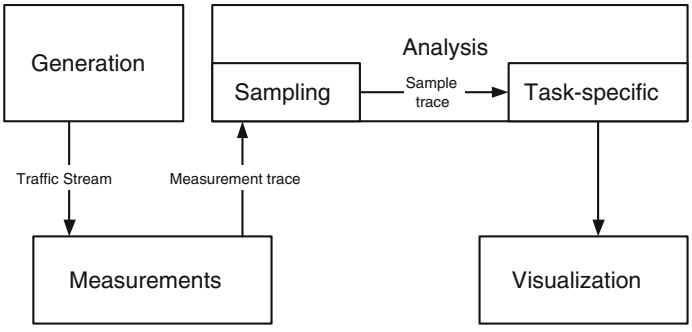

Fig. 1. Network performance measurement framework

in Figure 1. The generation module's task is to generate traffic. The measurement module captures and filters this and other traffic streams at one or multiple points in the network. There are no restrictions on which layers can be used by the generation and measurement modules, both can be done from the physical layer up to the application layer. The measurement module is so named to emphasize that it collects PDUs, measures time and does not perform any analysis. The analysis module processes the data provided from the measurement module, it does this by first sampling the data and then performing a task-specific operation, which is entirely dependent on the type of analysis that is to be performed. The output from the analysis module is then sent to the visualization module, that displays the results from the analysis. Using this framework, it is possible to clarify the semantics, detect and discuss error sources and allow for independent development of the modules. Each module has a specific role. In the following sections an overview will be given on the framework modules.

2.1 Generation

Generation deals with the construction of traffic according to a given set of parameters. It has mainly been used as a part of active measurements, but recently it has also been used together with passive measurements. Traffic generation can be performed at the same level as measurement, and hence it is subject to the same accuracy problems. Instead of detecting events, it generates events. The output from the module is a network traffic stream that is fed into a network and eventually the measurement module. With respect to ALM the generation module is interesting as a lot of ALMs are based on some externally generated data.

2.2 Measurement

The measurement module deals *only* with the collection and filtering of network traffic and associated parameters, i. e. no aggregation or parameter extraction takes place. Filtering is a process that determines if the collected data matches certain criteria, and if it does not the data is discarded. Measurements can be

done at various levels in a network, from the physical layer all the way up and into the application layer [3]. In many research publications there is a differentiation between active and passive measurements. Here, no such differentiation will be made since there is no difference between them in the measurement module. Both active and passive measurements use the measurement module to collect PDUs. The output from the measurement module is a *measurement trace*. The trace can be stored in a file or temporarily stored in memory.

2.3 Analysis

The analysis module deals with everything after measurement and prior to visualization, hence this is a very large module. It can be divided into two sub-modules; *sampling* and *task-specific*. The sampling process is used to interpret the measurement trace provided by the measurement module. There are two types of sampling, time-based and event-based, comparable to simulations with fixed-time-increment or event-based-increments [5]. Figure 2 shows the difference. In time-based sampling the PDUs of the measurement trace are arranged on a time line with markers at fixed intervals T_S time units apart. While for event-based sampling, the passing of T_S time units does not need to be the sample criteria, in Figure 2 the criteria is the arrival of a PDU. Time-based sampling can be seen as event-based sampling with the criteria to sample each T_S time unit. The output from the sampling process is a *sample trace*. One or more of these sampling processes can be applied in sequence, the first one operates on the measurement trace and the other operates on intermediate sample traces. One or more of these sample traces are then subject to the task-specific analysis. The task-specific sub-module can be anything from simple averaging to protocol or user behaviour analysis. Furthermore, it is not limited to using only one sample trace. The output from the module is analysis specific and preferably adjusted for the following visualization.

2.4 Visualization

The last module, visualization, presents the analysis results to the user. Since the visualization module is the only visible module, this module will have a profound impact on the interpretation of the results. This module can hide, emphasise or

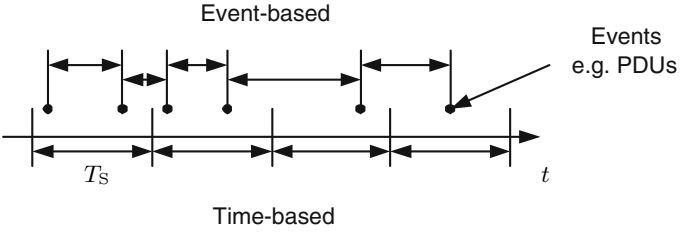

Fig. 2. Difference between time and event based sampling

distort the results obtained by the analysis module. For example, the visualization module can choose whether or not to display confidence intervals (if these are provided by the analysis module). Distortion occurs for instance, if a value is printed with too few digits. Just to mention a couple of examples: the text output from `ping` [6] in a console window and a topology map of a network [4] are both the results of this module.

3 Measurement

The measurement module only deals with the collection and filtering of PDUs and parameters associated with these. The result of the measurement module is called a measurement trace, in other works this is sometimes referred to as a packet trace. The trace can be virtual or physical; a physical trace is stored in a semi-permanent memory like a file, while a virtual or logical trace would only exist in memory. A trace can be as small as one PDU or contain millions of PDUs. The measurement trace is used to reduce the amount of data sent to the analysis module. The content of the measurement trace is in direct relation to the type of analysis that will be performed later on. For instance, some analysis methods are only interested in the PDU arrival times, others in PDU contents and some methods are interested in combinations of these fields.

3.1 Parameters

What parameters should the measurement trace contain? The PDU or at least some parts of it. The trace should also contain the collection location w of the PDU, this includes both where in the network stack (logical location) as well as where in the world (physically location) the PDU was collected. The trace should also include timing information such as when the PDU started to arrive T_{A} and when the PDU was completely received T_{E}. If the trace contains both time values, it will be possible to determine behaviour in environments with variable capacities since it is possible to calculate the capacity perceived by the PDU from its length and timing information. In addition to these four, two more values should be included the PDU length L and PDU capture length L_{C}. These parameters are listed in Table 1. In addition to these parameters a measurement trace should also have a set of meta-data. The meta-data can

Table 1. Measurement trace parameters

Name	Symbol	Description
PDU	p	The PDU, or parts of it
Location	w	Position of collection, logical and physical
Arrival time	T_{A}	Arrival time of the PDU's first bit/byte [s]
End time	T_{E}	End time of the PDU's last bit/byte [s]
Length	L	Length of the original PDU [bit]
Capture length	L_{C}	How much of the PDU is stored here [bit]

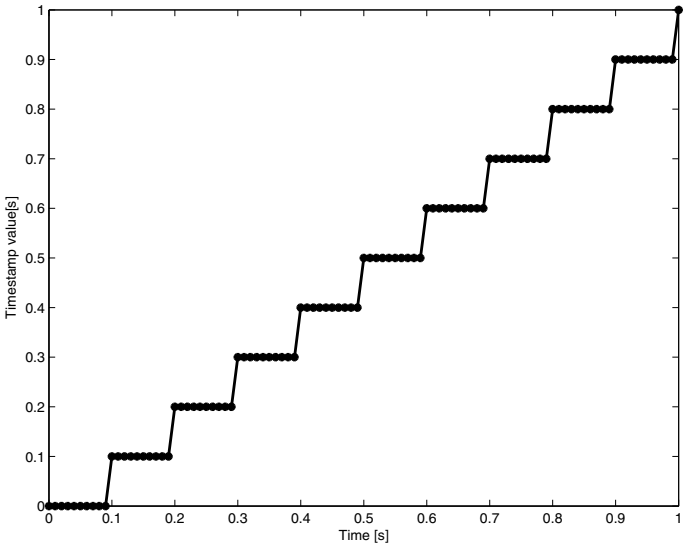

Fig. 3. Timestamp staircase

be filter information, network environment description, capture tools, hardware specifications, software versions etc. This information is however more static than the parameters that possibly change with each PDU. Hence, each measurement trace should be accompanied by its meta-data.

Timestamps. Recall that the PDU arrival time T_{A} identifies when a PDU started to arrive and PDU end time T_{E} identifies when the PDU was completely received. These values are usually refered to as timestamps. A timestamp is associated with a timestamp accuracy T_{Δ} provided by the measurement system including both hardware and software. A timestamp is obtained by reading a counter [8], which is updated at given intervals, and converting it into a time value. The length of these intervals determines the resolution of the timestamp and is the lower bound on the timestamp accuracy. To illustrate this, Figure 3 shows an artificial example of a timestamp sequence. The x-axis shows the true time and the y-axis the timestamp value. The staircase is created because the timestamp counter is not continuously updated; in this example it is updated every 0.1 s. In some cases, the timestamp counter is large enough to keep a smaller value than the update interval, for instance if the counter can support a timestamp with a resolution of 0.001 s. However, a high counter resolution does not increase the timestamp accuracy; it may however give false confidence in the values.

In [9] the author presents the following terminology regarding clocks: *Resolution* is defined as the smallest unit by which a clock is updated, also known as a *tick*. *Offset* specifies the difference between a particular clock and the *true* time as definied by national standards. *Skew* is the frequency difference between the clock and a national standard, or the first derivative of the offset at a particular moment, and *Drift* specifies the second derivative of the offset, or the variation of the skew.

The timestamp accuracy obtained is a combination of all these factors as well as processing delay and scheduling when collecting the timestamp. T_Δ indicates how accurate a timestamp is. The timestamp accuracy is a local value, meaning that unless two clocks are synchronized then their timestamp accuracy cannot be compared. However, if synchronization is applied this will be visible from the timestamp accuracy. Furthermore, the timestamp accuracy does not specify the offset of a timestamp. The true offset value is hard or impossible to include for every PDU, however information about the offset should be included in the meta-data associated with the measurement trace. The timestamp accuracy also includes information about the system that collected the timestamp, in the sense that it reflects the impact of the entire system and not only that of the clock.

Clock Synchronization. Clock synchronization is divided into two tasks; time synchronization and frequency synchronization. Time synchronization is used to give two separate clocks the same value, frequency synchronization is used to make the clocks tick at the same rate. For instance, let two clocks be time-synchronized at time zero. Wait a while, and then read the time values from both clocks simultaneously. Now the first clock might report 120 281 time-units (tu) and clock two reports 120 304 tu. On the other hand, if two clocks are frequency-synchronized but not time-synchronized, the initial time reading will produce two different values for example 1201 tu and 11 029 tu, and when the time is reread after a while the values might be 1450 tu and 11 278 tu. These clocks are frequency synchronized since the same time (249 tu) elapsed on both clocks. By having a common/public reference, it is possible to synchronize many clocks [10], [11], [12], [14].

Depending on how the clocks are synchronized, the timestamp accuracy is affected. Time synchronization involves changing the counter value, this can cause jumps in time, either forward or backward. If such a jump occurs during a measurement the measurement section involving the time correction cannot be used. For this reason, if time-synchronized measurements are required the devices should be synchronized prior to starting the measurement.

Frequency synchronization is on the other hand a continuous process. It usually operates by modifying a variable v which is used to create a synthetic clock frequency S_f. The variable describes the relationship between the crystal frequency C_f and the desired synthetic frequency: $S_\mathrm{f} = f(C_\mathrm{f}, v)$. The synthetic frequency is then used to update the time counter in a more stable way than if the crystal frequency would be used directly. This is needed since the crystal frequency changes with age and temperature. When a crystal is powered on, its frequency can vary significantly. Thus, before a measurement is started the crystal needs to reach its operating temperature. At this point the synchronization method should be applied, and once the crystal frequency has become stable the measurement can begin. Depending on the equipment and environment, this time can vary significantly but as a rule of thumb, 15–30 minutes should be sufficient to obtain crystal frequency stability [13].

The most common way to synchronize computers on the Internet is the Network Time Protocol, NTP [15], [11]. Since NTP is used so widely, it is interesting

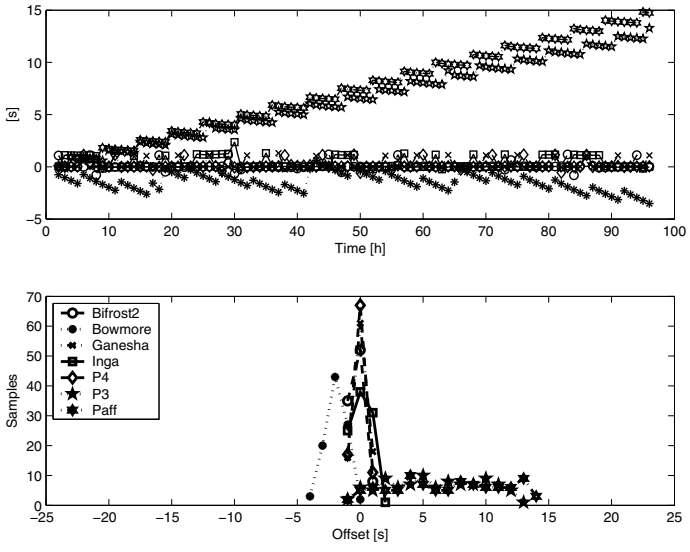

Fig. 4. Clock offset, with or without NTP synchronization

to see how it conditions a computer's clock [16]. Figure 4 shows the time offset of five computers compared to a common reference, each hour the syste time was compared to the refence and logged to a file. The top graph shows the time series and the bottom graph shows the corresponding histogram. Here it is obvious that both P3 and Paff are unsynchronized despite the NTP daemon being started on the P3 and no NTP related errors being detected in any of the logs found on the machine. Bowmore deviates from the others since it shows a negative offset, i.e., it runs slower than the NTP reference. The difference also seems to be growing, Bowmore was synchronized not by running the NTP daemon, but by issuing `ntpdate` once every 24 hours. This command will correctly synchronize the time, but will not correct the frequency. This is visible in the trace, even though the offset is reduced after 18, 42 and 66 hours, there is a drift in the behaviour. A reduction was also expected around hour 90, but this seems to be missing, causing Bowmore to be almost 5 seconds behind the NTP reference. This behaviour is emphasised by the histogram, where the Bowmore's shape has a small tendency to become wider. The remaining four devices: Bifrost2, Inga, Ganesha and P4 are synchronized within -2 to $+2$ seconds.

Timestamping Methods. When collecting timestamps in software there are two primary methods that are used; the Timestamp Counter (TSC) [17] and Get Time of Day (GTOD). The TSC reads the CPUs internal clock counter, usually via an assembler call, while the GTOD uses a system call `gettimeofday` to obtain a time value. The benefit with GTOD is that it reports the time directly, while the TSC reports a counter that represents the number of CPU cycles since the computer was started, with one cycle completed approximately

every $1/f_{\mathrm{CPU}}$, where f_{CPU} denotes the CPU clock rate. This value has to be divided by f_{CPU} to get a time value. A problem is that the actual cycle time depends on the crystal frequency, hence it is subject to aging, heat and many other sources of errors. Effectively one has to estimate the CPU speed over some interval, preferably determined by some external time source. The TSC method has a clock resolution of $1/f_{\mathrm{CPU}}$ which can be quite high, while the resolution for the GTOD method is determined by the operating systems clock, usually in the order of a few μs.

The TSC method enables higher resolutions, < 1 ns for $f_{\mathrm{CPU}} > 1$ GHz, and should as such be used. However, the method does come with a set of problems. The conversion from CPU cycles to time is a problem that needs to be addressed and solved. Related to this is synchronization, see [17] the authors they discuss synchronization methods in detail. A third problem that both methods have is the operating system's scheduling. The problem is present for all processes in a multi-tasking system, all user processes can be paused in their execution. If this happens, then regardless of clock method the results will be compromised. An ideal solution is to use the TSC method in combination with code that is executed by the kernel, for instance the network driver [17,18] where scheduling effects can be minimized.

If PDUs are collected in the lower layers of the stack, the impact that both system and stack have on them is minimized. Furthermore, if the TSC approach is used, the PDU timestamps can be quite accurate. However, if measurements are performed at the upper layers, i. e., at the application level, then both stack and system need to be evaluated since the behaviour that is observed is a combination of the network, network stack and system. Hence, conclusions drawn from this data must account for this. Ideally, application level measurements should be backed up with measurements at the physical or link layer to monitor the input to the stack.

As stated before, the location parameter is important and when performing stack or application measurements it is crucial to specify where the PDUs are collected. At first glance it seems obvious that they should only collect the location parameter after the PDU has been obtained, for instance after the `read` command has returned. But by adding a second timestamp before the `read` command, a lot more can be done. It is possible to determine the processing time of the `read` command and indirectly see if there was any buffering in it and the per-PDU processing time can also be evaluated. On the other hand, by adding a second timestamp, more data is created and requires more processing power. It can also be argued that this is evaluation of the system and not of the network.

Another problem is timestamping location. For example, in Linux it is quite easy to access the raw data that is passed from the link layer to the network layer. Now if the application timestamps the PDUs, this is an application layer timestamp, not a link-layer timestamp. In this case to get a link-layer timestamp the kernel has to be modified.

PDU Location. The location parameter is rarely discussed, but it is very important. The location parameter identifies where in the network stack (logical

location) and where in the world (physical location) the parameters were collected. One of the reasons that this is rarely mentioned in publications is that the location is usually known to those that perform the measurement and it is not needed to motivate the end results. However it is important to remember and it becomes even more important when comparing measurements at different physical and logical locations. The physical location could for instance be specified using the GPS coordinate system. To determine the logical location can be somewhat more problematic. For example, assume that the logical location identifies a particular layer in the OSI stack. If the logical location is stated as the data link, does this mean the interface towards the physical layer, the network layer or everything in-between? This needs clarification, i.e., saying that the measurements were performed on the data link layer is insufficient. Furthermore, if the timestamping is not performed at the PDU collection location, then there will be a difference between the timing information and the PDU contents. This can at worst cause problems, and atleast confusion.

4 Sampling and Analysis

Once a measurement trace has been obtained, the next step is to analyse it. This is the task of the analysis module, it can range from simple parameter extraction, averaging to modelling of user or application behaviour. Common to all of them is the need to sample the measurement trace. The sampling can be seen as a sub-module of the analysis module. There are two types of sampling; time-based sampling and event-based sampling. The result from the sampling process is a sample trace, which is delivered to the task specific sub-module. The analysis can be performed both in the time domain or in the frequency domain. On top of this, the scaling behaviour can be analysed on different timescales [19].

4.1 Sampling

Sampling describes the process of converting a measurement trace into a format suitable for the subsequent analysis. In its simplest form the sampling process can be a format conversion, i. e., converting a Unix timestamp to a human readable format, or it can involve filtering and simple arithmetics [29]. The sampling process can be done in one operation or a sequence of operations. The two ways of sampling a measurement trace are denoted time-based or event-based, which are comparable to the two approaches that can be used in simulations; fixed-increment time advance or next-event time advance [5].

Sampling differs from the classical signal-processing approach, where the sample instance indicates that a value is to be read from an A/D-converter. The sampling here is more of an evaluation of the current conditions and it can involve simple arithmetics, examples for which will be provided below.

Time-based Sampling. Time-based sampling is the classical procedure for sampling a signal. Given a measurement trace D that contains three parameters; PDU arrival time $T_{\mathrm{A},i}$, PDU length L_i and the PDU p_i. These are then placed

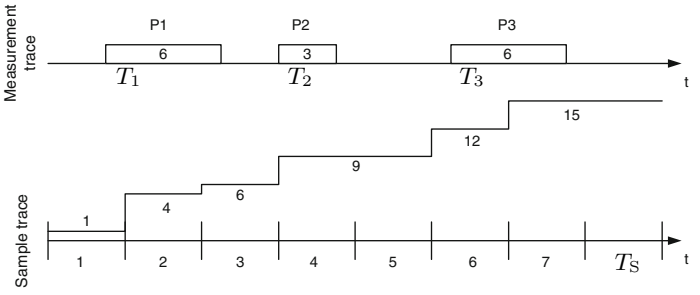

Fig. 5. Time-based sampling, data counter

on a timeline, with markers T_S time units in-between. Within each of these intervals one or more of the parameters are aggregated and the result used in the following analysis. In Figure 5 a simple example is given. The measurement trace is sampled each T_S time unit, at which the total amount of data received up to and including the interval is written to the sample trace. This is quite a simple operation, a slightly more complicated operation would be to sample the amount of data received in the latest interval, since it would involve resetting the counter after each sample interval.

Event-based Sampling. Event-based or adaptive sampling does not necessarily use time as the sample criteria. For instance, the reception of n PDUs can be the criteria for sampling, or that T seconds of silence has passed since the last received frame. However, regardless of what sample criteria is used, the aggregation is done in the same way as in time-based sampling. Using the same measurement trace as before, the resulting event-based sample trace is shown in Figure 6.

Combination and Sequence of Sampling. It is quite common to use a combination of sampling techniques, which are applied in sequence. The first process is applied to the measurement trace, the second to the sample trace produced, the third to the second sample trace and so on [20].

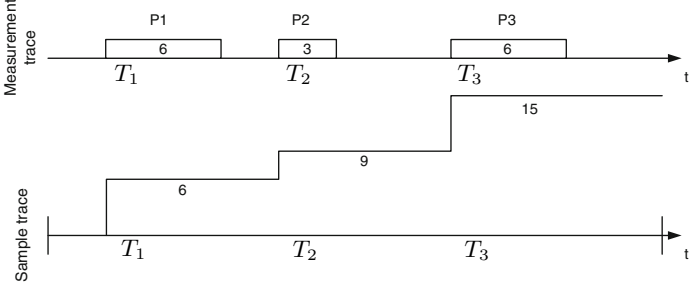

Fig. 6. Event-based sampling, bitrate

Here, a simple notation is introduced, the sampling techniques are listed in the sequence that they are applied. Time-event-based sampling means that the measurement trace was sampled using time-based sampling, and the intermediate trace was then sampled using an event-based criteria. For example, a periodical SNMP query would be a time-time-based sampling, if the SNMP agent internally used time-based sampling of the counters within the device [21]. If the SNMP agent internally used event-based sampling, the correct notation would be event-time-based sampling. An event-event-based sampling could describe a tool that first calculates the PDU inter-arrival time, followed by the inter-arrival time between two PDUs. Time-time-based could be a scaling analysis, like the one performed in [19]. The sample criteria should be supplied in the meta-data associated with a sample trace, this also includes the meta-data from the measurement trace.

4.2 Analyser and Software Impact Numerical Precision

All computers keep time by counting the number of seconds that has passed since a particular time instance, in the majority of systems this is the time since 1970-01-01[1]. At the time of writing the number of seconds that has passed is 1 256 871 240 (2009-10-29 00:00:00). Depending on how this value is represented, it will eventually wrap around and become zero again. These timestamps are stored using a fixed number of bits. For a 32-bit representation the counter will wrap to 0 around 2038-01-19. But this number only holds the seconds, not any fractions of seconds.

For this reason a timestamp is usually divided into two numbers, one for the second and another for the fractional second. When this data is read into an analyser, it might be combined into a single value for simpler processing. Here the problems arise from the limited accuracy in computers. If a large value and a very small value are added together, the new number might drop some of the digits in the smaller number in order to correctly represent the larger number. If the numbers would be kept separate, then the number of operations needed to handle them would (at least) double.

In a computer a float value is represented as two numbers, the exponent and the mantissa, and to further complicate things it is represented in a binary system (base-2) and not with a decimal system (base-10) as we are accustomed to. The mantissa stores the number and the exponent a scaling factor. For example, when storing the value 4049 (base-10) in a system with an 8-bit mantissa and a 4-bit exponent. In a computer this is represented as $N = 0.988\,281\,25 \times 2^{12}$, where the mantissa is 0.988 281 25 and the exponent is 12. Here, the number N does not represent 4049, but 4048 since this is the closest value that can be represented with an 8-bit mantissa. By increasing the mantissa size a better representation of the value can be obtained. Increasing the size to 16 bits, 4049 can be represented correctly.

Should one wish to represent a timestamp as one single value, including both seconds and fractional seconds, then one needs to bear in mind that the analyser's

[1] YYYY-MM-DD.

or computer's representation prefer large numbers. That is, even if a timestamp has an accuracy of 100 ns, combining the second and fractional second values may cause a loss of accuracy. To evaluate this for some common analysis software, a small test was created. The test was performed by adding two values, one represented the number of seconds since a given reference and the other represented a number of fractional seconds. The system was then requested to print the new value using its maximum resolution.

In Tables 2 and 3 a comparison between four software systems (Matlab, R, Perl and Python) and three number representations in C++ (`double`, `long double` and `quad double` [22]) is shown. The first column holds the reference date and the second column contains the number of seconds that have elapsed since the reference date. The third column holds the fractional seconds. The values in the second and third columns are added to create a new value x, which is in turn printed by the systems listed in columns four to six. Starting with Table 2, it is clear that if the entire second count since 1970 is kept, one can only be sure that the ten-μs digit is correct. If the reference is changed to 2000-01-01, Matlab and R can be trusted to the one-μs digit. By choosing an even closer reference, 2005-01-01, one may expect to be able to rely on the 100 ns value, this is not the case. But by choosing a reference only a week away, one can trust the value representing 100 ps, and by choosing a reference one day away the smallest number one can rely on is 10 ps. Worth noting here is that if one wishes to have a μs accuracy then one must use 2000-01-01 as the time reference instead of the default 1970 reference. A second comment is that if one performs a measurement that spans a week it is possible to obtain 10 ps if the first day of the measurement is used as a reference. If 1970-01-01 was used as a reference one will only obtain 10 μs.

In Table 3 the output from a C++ program is shown, here if x is stored as a float with `double` precision and 1970 is used as a reference then the representation is quite accurate, and if the fractional is decreased to 0.1 μs then the value is not correctly represented. When x was represented as a `long double` then the value is correctly identified and the same is true when the fractional was only 0.1 μs . The output from the `quad double` representation, when using 1970 as the reference date is not as accurate as the `long double` representation. If one uses a `double` to represent timestamps and these are accurate to one μs , one must use 2000-01-01 as the reference date. If the timestamps are accurate to the nanosecond, then one must use a reference that is less than 24 hours away, and should not compare values that are more than 24 hours apart. For the `long double` representation things look much better, in fact it has the best representation of x for all reference dates and fractional values. The `quad double` representation is almost as accurate as the `long double`, since it seems to be rounding the values differently.

4.3 Task-Specific Analysis

Based on the sample trace, a multitude of different analysis methods are available, however, there are far to many to discuss in the context of this thesis.

Table 2. Matlab, R and Perl accuracy

Reference		Environment		
Seconds	**Fractional**	**Matlab 6.5**	**R 2.1.0**	**Perl 5.8.2 (windows) Python 2.5.3 (windows)**
1970-01-01				
1116885600	1e-6	1116885600.0000010	1116885600.000001	1116885600.000000095367432
1116885600	1e-7	1116885600	1116885600	1116885600
2000-01-01				
170200800	1e-6	170200800.000001	170200800.00000101	170200800.000001013278961
170200800	1e-7	170200800.000000090	170200800.00000009	170200800.000000089406967
2005-01-01				
12348000	1e-6	12348000.000001	12348000.000001	12348000.0000010002404451
12348000	1e-7	12348000.0000001010	12348000.000000101	12348000.0000001005828381
2005-05-17				
604800	1e-6	604800.00000100001	604800.0000010000	604800.000001000007614493
604800	1e-7	604800.0000001	604800.0000001	604800.000001000007614493
604800	1e-10	604800.00000000012	604800.00000000012	604800.000000000116415322
604800	1e-11	604800	604800.00000000000	604800
2005-05-23				
86400	1e-6	86400.000000999993	86400.000000999993	86400.000000999930625781
86400	1e-7	86400.0000001	86400.000000100001	86400.000000999930625781
86400	1e-11	86400.000000000015	86400.000000000015	86400.000000000145519152
86400	1e-12	86400	86400	86400
2005-05-24				
3600	1e-12	3600.0000000000009	3600.0000000000009	3600.0000000000009094947

Table 3. C++ accuracy

Reference		Environment		
Seconds	**Fractional**	double	long double	quad double
1970-01-01				
1116885600	1e-6	1116885600.0000009536743..	1116885600.00000100000.....	1116885600.00000092.....
1116885600	1e-7	1116885600	1116885600.00000100000.....	1116885600.00000003.....
2000-01-01				
170200800	1e-6	170200800.0000010132789..	170200800.000000999993.....	170200800.000000995.....
170200800	1e-7	170200800.0000000894069..	170200800.000000100000.....	170200800.000000107.....
2005-01-01				
12348000	1e-6	12348000.0000010002404...	12348000.000001000000.....	12348000.0000010003.....
12348000	1e-7	12348000.0000001005828...	12348000.000000099999.....	12348000.0000000988.....
2005-05-17				
604800	1e-6	604800.0000010000076...	604800.000000999999.....	604800.000001000004.....
604800	1e-9	604800.0000000010477...	604800.000000000099998...	604800.000000000981.....
2005-05-23				
86400	1e-6	86400.0000009999930...	86400.000000999999997..	86400.0000010000057....
86400	1e-9	86400.0000000010040...	86400.000000000999996..	86400.0000000010004....
86400	1e-12	86400	86400.000000000001001..	86400.0000000000056....
2005-05-24				
3600	1e-12	3600.0000000000009...	3600.000000000001000..	3600.00000000000097...

What needs to be pointed out is that the analysis might emphasize the errors accumulated in the sample trace. It is easy to believe that the error can be reduced by increasing the amount of data, i. e,. by collecting 100 000 samples instead of 10 000 samples. However, the error in each of these samples is independent on the number of samples collected.

In Figure 7 an example of a measurement trace is shown, consisting of three PDUs 4, 3 and 6 bytes long, it is subject to a time-based non-fractional PDU accounting sampling that calculates the bitrate. The resulting sample trace is denoted as vector $\mathbf{V}_{\mathrm{st}}$ and contains realisations of the random variable V_{st}, given in bps. V_{st} contains three non-zero values:

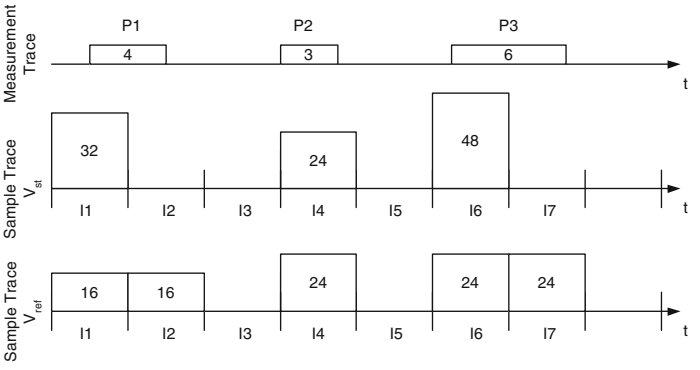

Fig. 7. Analysis problem

$$\mathbf{V}_{\mathrm{st}} = [32, 0, 0, 24, 0, 48, 0] \quad \mathbf{E}[V_{\mathrm{st}}] = 14.86 \quad \mathbf{Var}[V_{\mathrm{st}}] = 393$$

let us compare V_{st} to a sample trace $\mathbf{V}_{\mathrm{ref}}$ that accounted for the fractional PDUs. $\mathbf{V}_{\mathrm{ref}}$ would contain five non-zero samples:

$$\mathbf{V}_{\mathrm{ref}} = [16, 16, 0, 24, 0, 24, 24] \quad \mathbf{E}[V_{\mathrm{ref}}] = 14.86 \quad \mathbf{Var}[V_{\mathrm{ref}}] = 116$$

Now, comparing the mean values of these vectors shows that they are identical, but their higher order statistics differ significantly. However, the last zero sample in $\mathbf{V}_{\mathrm{st}}$ is necessary to achieve the same mean values. For instance, if both traces were such that interval I7 was not included, the statistics would be different.

$$\begin{aligned} \mathbf{V}_{\mathrm{st}} &= [32, 0, 0, 24, 0, 48] \quad \mathbf{E}[V_{\mathrm{st}}] = 17.3 \quad \mathbf{Var}[V_{\mathrm{st}}] = 420 \\ \mathbf{V}_{\mathrm{ref}} &= [16, 16, 0, 24, 0, 24] \quad \mathbf{E}[V_{\mathrm{ref}}] = 13.3 \quad \mathbf{Var}[V_{\mathrm{ref}}] = 119 \end{aligned}$$

It is therefore important to cover all intervals in which PDUs are supposed to be present. Especially if fractional PDUs are not taken into account, extra sample intervals might need to be added.

To further complicate matters, the time-of-day comes into play, as the network behaviour is influenced by the time-of-day. The same reasoning can be applied for the problems caused by the timestamp accuracy. In general one should assume the worst case scenario, where errors enhance each other after each step of processing. Thus, the best way to go is to perform an error analysis for the entire system and analysis method, in order to determine the quality of the final results.

5 Application Level Measurements

Application level measurements collect PDUs at, or above, the application layer in the network stack [23]. These are usually collected using regular user applications that are executed and scheduled in the user domain by the operating

system. These measurements are then used to draw conclusions about the network behaviour. However, the results are not only affected by the network, but also by the hardware, software and in particular the operating system of the computer that performs the measurement. Hence it is necessary to investigate the influence these components have on the measurement results [24].

We will do this by using the timings inbetween packets, i.e. the Inter-Packet Time (IPT), as this should not be changed as long as the packet passes either up or down the stack. However, this holds only if the stack is not congested. If it is then the packets may be delayed, or even buffered before deliverd to the next layer in the stack. This can cause the IPT to either shrink or grow. Many tools have been built indirectly based on this assumption. They usually consist of two parts, a sender and a receiver. The sender is configured to send a packet, pause execution for some time, and then repeat the procedure until a predefined number of packets has been transmitted. The receiving side will then receive the packets and calculate the IPT and compare it to the user defined IPT at the sender. This works fine, if you know that the sender really behaves as desired.

5.1 Setup

We evaluated three different ALM tools; the first (A) was implemented in classical C, the second (B) in Java and the third (C) used C#. Furthermore, the C and C# implementations use UDP for communications, while the Java application uses TCP. The setup is shown in Figure 8. To collect the network data, we use the Distributed Passive Measurement Infrastructure [25]. The PDU are copied by the wiretaps and sent to the MP, were we use DAG3.5E cards [26] synchronized using GPS, to collect them. The hosts (H1 and H2) were identical in terms of hardware for each of the experiments. For tool A it were Pentium-4 2.8 GHz systems with 1 GB of RAM and a built-in 1000Base-TX (configured to operate at 100 Mbps) cards. Tool B and C used Pentium-3 667 MHz, 256 MB RAM and built-in 100Base-TX cards. For Tool A the operating system was a Linux 2.6 system, while for B and C it was Windows XP (SP2). For B the Java version was 1.5.0, and for C the .NET framework was 2.0. The evaluation was done by having host H1 and H2 running the tools and collecting the application-level traces, while the DPMI collected the link layer traces. The data was then analyzed offline using Matlab.

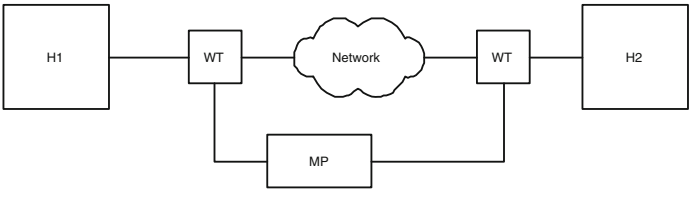

Fig. 8. Setup used to evaluate ALM

All three tools consists of two parts a sender and a receiver, the sender is located on H1, and is configured to transmit its data to H2, were the receiving part resides. The senders are configurable with respect to load into the network, this is done via controlling the inter packet time (IPT), the payload size and the number of packets to send. The receiver applications timestamps the data arrival, and stores these in a static vector that is written to file after the experiment has been completed. Tool A and C used the TSC timestamping method, while B used the GTOD method. For Tool A the sender was configured to send 1472 bytes UDP datagram, corresponding to 1514 bytes at the link layer) once every 1 ms, and for B and C they sent 526 or 538 bytes, corresponding to 576 bytes at the link layer.

5.2 Analysis

To evaluate the quality of the ALM is to estimate the timestamp accuracy error, for details see [28]. Let $T_{x,y}(k)$ be the timestamp obtained at party x at layer y for PDU $k \in (1 \ldots n-1)$. Party x can either be the sender (s) or the receiver (r), and a layer y can be the application (a) or the link (l) layer. Let $IPT_{\mathrm{x,y}}(k, k+1)$ be an IPT for a PDU pair $(k,k+1)$ and $\epsilon_{k,k+1}$ a timestamp accuracy error for this pair, then T_Δ is obtained using:

$$\begin{aligned} IPT_{\mathrm{x,y}}(k, k+1) &= T_{\mathrm{x,y}}(k+1) - T_{\mathrm{x,y}}(k) \\ \epsilon_{k,k+1} &= IPT_{\mathrm{a,r}}(k, k+1) - IPT_{\mathrm{l,r}}(k, k+1) \\ T_\Delta &= |max(\epsilon_{k,k+1})| + |min(\epsilon_{k,k+1})| \quad \forall k \end{aligned}$$

Here we'll use a simplified method, were we only compare the statistics (mean and standard deviation) and the minimum and maximum values of the IPT.

5.3 Results

The results are summarised in Table 4. Remember that Tool A has a target of 1 ms, while for B and C the target is 125 ms. Looking at tool A, we observe that the difference between link and application is quite small for all the values. The extreme values are 60-70 μs different than the corresponding link values. Looking on Tool B this is quite different, here the minimum value is 90 ms smaller than the link value, and the maximum is 4 ms larger. For Tool C this looks better,

Table 4. Tools A–C: IPT's statistics at receiver

	Tool A		**Tool B**		**Tool C**	
Param.	Link [ms]	App [ms]	Link [ms]	App [ms]	Link [ms]	Appl. [ms]
min	0.44	0.37	109.96	20.00	65.98	65.97
max	1.56	1.62	236.92	241.00	184.94	184.94
mean	0.99	0.99	125.43	125.43	125.00	125.00
std.dev	0.01	0.02	1.23	5.33	0.75	0.75

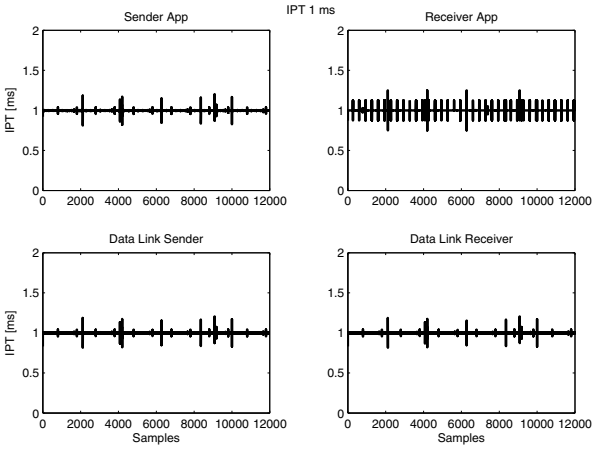

Fig. 9. Tool A: measured IPT at receiver for nominal IPT 1 ms

with only a small 1 μs difference. However, we need to remember that the Tool C had a load of 6 packets/s, while Tool A had a load of 1000 packets/s. Turning our attention to the statistics, all three tools match their target mean quite well. But, again Tool B has difficults, this time with the standard deviation, its significantly larger than that of the link layer. The main reason for this is that the GOTD of Tool B uses *System.currentTimeMillis()*, which had a resolution of 10 ms. As Tool A and C uses the TSC method they do not suffer from this problem, however they have other problems that we'll come to later.

In Figure 9–11 we show the IPT trace for tools. The top left graph shows the IPT at the sender at the application level, the graph below (bottom, left) shows the IPT at the link layer at the sources. Then the bottom right graph shows the IPT at the receiver, and then the top-right graph shows the IPT at the application layer of the receiver. Looking at Tool A (Figure 9 there is not much variability in the IPT data from sender to the data link receiver, but at the receiver application layer there is a slightly higher variability. For Tool B, shown in Figure 10 the results are quite different, here the IPT has a significant variability at both sender and receiver application, this is coupled to the timestamp resolution offered by Java. However, its interesting to note that at the link layer the IPT is not suffering from this. So, in Java its possible to execute a sleep that is smaller than 10 ms, but the reported time elapsed will be either 0 or 10 ms. For Tool C, the data is similar to Tool A. At both sides the IPT is more or less identical at both application and link layer.

In Figure 12 we see 20 IPT samples for Tool A. For Tool A, first we note that the correlation between the link IPT and the app IPT. Secondly, the app IPT seems to be a lot smoother. Initially, it's tempting to account this smooth behavior to the variability of the CPU frequency, and that the tool used the average CPU frequency that covered the entire experiment. Now as we do not have intermediate timestamps from the GTOD, we cannot recalculate the CPU

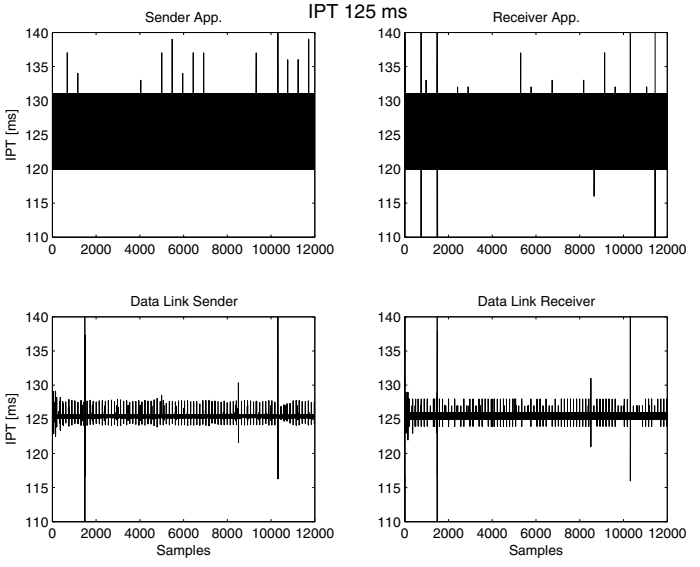

Fig. 10. Tool B: measured IPT at sender and receiver for nominal IPT 125 ms

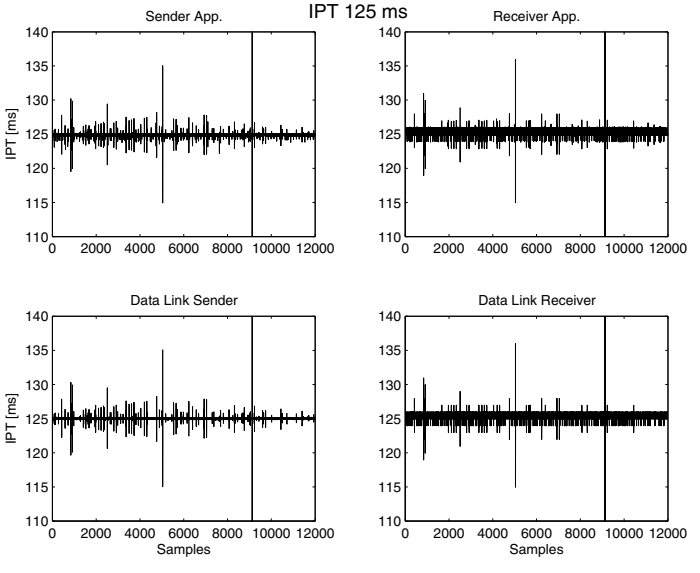

Fig. 11. Tool C: Measured IPT at sender and receiver for nominal IPT 125 ms

frequency for different periods. Thus, we cannot really explain why the application IPT is smoother, nor why the obvious peak and valley, at sample 94 and 95, does not show in the application IPT.

To investigate this further, we used the arrival time of the packets, and created a time trace relative to the arrival time of the first packet, for both link and

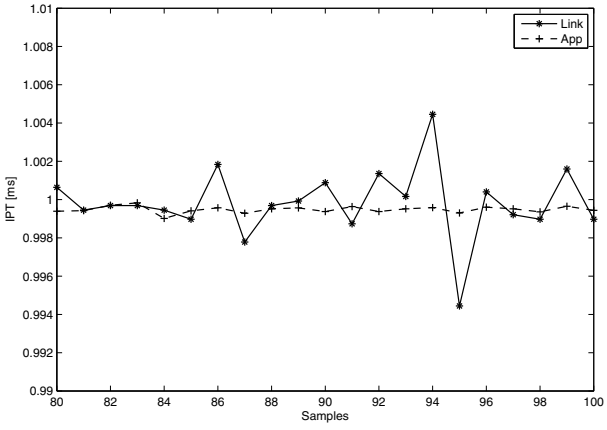

Fig. 12. Tool A: Detailed IPT 20 samples

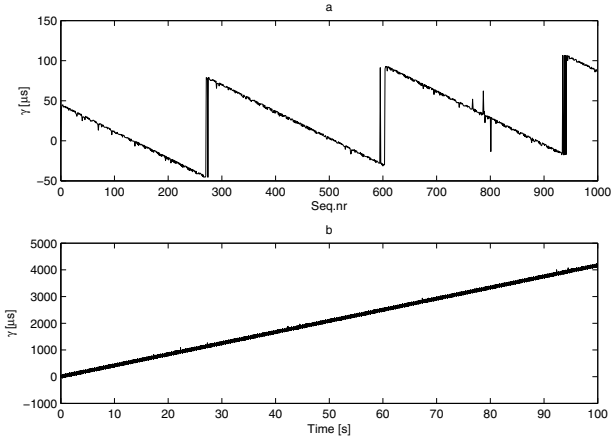

Fig. 13. Tool A: Difference between application packet arrival times and link layer arrival times

application layer. In the next stage we then created the difference between these time traces, like:

$$\gamma(k) = \hat{T}_a(k) - \hat{T}_l(k) = (T_{r,a}(k) - T_{r,a}(0)) - (T_{r,l}(k) - T_{r,l}(0) \quad (1)$$

As we know that the link layer timestamps are obtained from a properly synchronized source (DAG card + GPS), we can trust these. So, ideally γ should be very very close to zero. As this would indicate that the application layer timestamps are also obtained from a well behaved clock source. The results are shown in Figure 13. The upper graph (a), shows the first 1000 packets, corresonding to the first second. The sawtooth behaviour is obvious, this is a classical view of a clock that drifts. It seems that something tries to correct the clock every

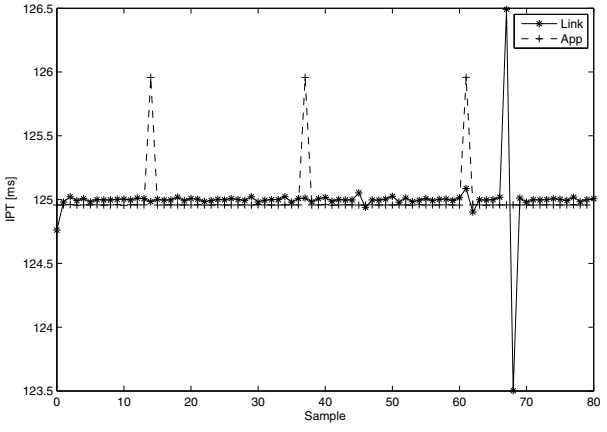

Fig. 14. Tool C: Details IPT for 80 samples

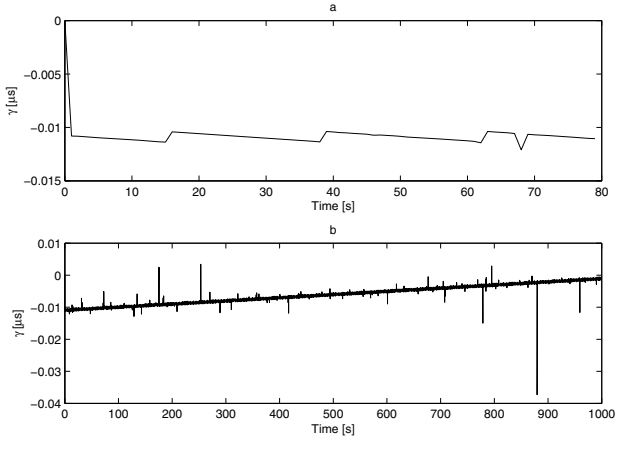

Fig. 15. Tool C: Difference between application packet arrival times and link layer arrival times

300 ms, but it over compensates, hence the clock also deviates even more as time goes. This is clearly visible in the lower graph (b), where we show the difference increases to 4 ms during a 100 second period. This corresponds to a drift of 3.5 s during one day, which is seriouly bad.

Turning our attention to Tool C, we show the a detailed view for 80 IPT samples in Figure 14. First we notice a periodic behaviour, every 23-24 samples. Secondly, the app IPT is consequently lower than the link IPT, not counting the periodic behavior. The increase/decrease indicated by the link at sample 67 passes totally undetected. We repeated the γ evaluation, and the results are shown in Figure 15. Graph a shows the first 80 samples, corresponding to

approximately 10 seconds of data. Again we see a sawtooth, however much weaker, but significantly stronger as well, as the scale is in ms, not μs . If when look on the lower graph, the drift is obvious, it goes from a -10 ms to 0 over 1000 s, thus during a day this system drifts 0.864 seconds. This is four times better than that of the four times faster the Pentium-4 system used by tool A.

6 Conclusions

In this paper we described a frame work usefull when discussing network performance. As ALMs are usually conducted with the intention of detecting network performance, the framework is also usefull in this context. Base on the framework we described the associated modules, and the problems associated with each module when it comes to the accuracy of measurements.

We showed how good timestamps can be destroyed by improper use of analysis tools.

We evaluated three ALM tools, one C++, one Java and one C# tool. We showed that all three generated statistical values that lookes similar to those obtained from the link layer. The Java application showed a high standard deviation, due to that the clock that was used to obtain the timestamp seemed to update in steps of 10 ms. Then we investigated the C++/C# tools, that both used the TSC method to obtain timestamps, we detected that both time traces exhibited clock drifts. The drifts seemed to be coupled to the CPU speed of the reciving host, and in this case a 2.8 GHz CPU generated a clock drift of around 4 ms in 100 seconds, while the slower CPU drifted around 1 ms in 100 seconds.

Based on this, if you choose to use the TSC method to obtain timestamps, make sure that you obtain a GTOD timestamp on a regular interval, preferably four times a second (cf. the conditioning behaviour shown in Figure 13 every 300 ms). This will allow you to condition your estimate of the CPU frequency, better, when you make the count to time conversion. The of course, this requires that the system clock is properly conditioned by some other means, either NTP or GPS.

Regardless, where you are measuring you should perform some steps before you can report results obtained from measurements. First you need to identify the parameter(s) and the desired accuracy of these parameters. Then evaluate the accuracy that your HW/SW combination delivers, and if needed replace parts. Do a test where you evaluate the system over the intended measurement period. Evaluate the accuracy that your SW/analysis tool gives, the best way it to use artificial data that gives you full control on the desired output. Perform an error analysis, were you estimate the worst case error obtained in one sample. Then do your measurements, and report that you "measured X, and got $X \pm Y$. Also make sure that your systems are synchronized to a well known reference, and ideally the time should be traceable.

References

1. Schormans, J.A., Timotijevic, T.: Evaluating the Accuracy of Active Measurement of Delay and Loss in Packet Networks. In: MMNS (2003)
2. Chevul, S., Isaksson, L., Fiedler, M., Karlsson, J., Lindberg, P.: Measurement of application-perceived throughput of an E2E VPN connection using a GPRS network. In: 2nd EuroNGI IA.8.3 Workshop (2005)
3. Open Systems Interconnect - Basic Reference Model, Recommendation X.200 (1994)
4. CAIDA, `http://www.caida.org/tools/measurement/skitter/`
5. Law, A.M., Kelton, W.D.: Simulation, Modelling and Analysis. McGraw-Hill, New York (1991)
6. Muuss, M.: The Story of the PING program, `http://ftp.arl.mil/~mike/ping.html`
7. Bureau International des Poids et Mesures, `http://www.bipm.org/`
8. Donnelly, S.: High Precision Timeing in Passive Measurements of Data Networks. Phd Thesis, The University of Waikato (2002)
9. Paxson, V.: On calibrating measurements of packet transit times. SIGMETRICS Perform. Eval. Rev. (1998)
10. Skeie, T., Johannessen, S., Holmeide, Ø.: Highly Accurate Time Synchronization over Switched Ethernet. In: Proceedings of 8th IEEE conference on Emerging Technologies and Factory Automation, ETFA (2001)
11. Mills, D.L.: Improved algorithms for synchronizing computer network clocks. IEEE/ACM Transactions on Networking (1995)
12. Zhang, L., Liu, Z., Honghui Xia, C.: Clock synchronization algorithms for network measurements. In: INFOCOM (2002)
13. Dietz, M.A., Ellis, C.S., Frank Starmer, C.: Clock Instability and Its Effect on Time Intervals in Performance Studies. Technical report DUKE–TR–1995–13 (1995)
14. Wang, J., Zhou, M., Zhou, H.: Clock synchronization for internet measurements: a clustering algorithm. Computer Networks (2004)
15. Mills, D.L.: RFC1305:Network Time Protocol (Version 3): Specification, Implementation and Analysis. IETF (1992)
16. Smotlacha, V.: Experience with precise timekeeping in end-hosts. CESNET Technical Report 18/2004, `http://www.ces.net/project/qosip/`
17. Veitch, D., Babu, S., Pásztor, A.: Robust Synchronization of Software Clocks Across the Internet. In: Proceedings of the Internet Measurement Conference (2004)
18. Deri, L.: nCap: Wire-speed Packet Capture and Transmission. In: E2EMON (2005)
19. Carlsson, P.: Multi-Timescale Modelling of Ethernet Traffic. Licentiate Thesis, Blekinge Institute of Technology (2003)
20. Claffy, K.C., Polyzos, G.C., Braun, H.-W.: Application of Sampling Methodologies to Network Traffic Characterization. In: SigComm (1993)
21. Carlsson, P., Fiedler, M., Tutschku, K., Chevul, S., Nilsson, A.: Obtaining Reliable Bit Rate Measurements in SNMP-Managed Networks. In: Proceedings of the 15th ITC Specialist Seminar (2002)
22. High-Precision Software Directory, `http://crd.lbl.gov/~dhbailey/mpdist/`
23. Feng, W.-C., Gardner, M.K., Hay, J.R.: The MAGNeT Toolkit: Design, Implementation and Evaluation. Journal of Supercomputing (2002)
24. Danzig, P.B.: An analytical model of operating system protocol processing including effects of multiprogramming. SIGMETRICS Perform. Eval. Rev. (1991)

25. Arlos, P., Fiedler, M., Nilsson, A.A.: A Distributed Passive Measurement Infrastructure. In: Proceedings of Passive and Active Measurement Workshop (2005)
26. Endace, http://www.endace.com
27. Arlos, P.: On the Quality of Computer Network Measurements. Phd. Thesis, Blekinge Institute of Technology (2005)
28. Arlos, P., Fiedler, M.: A Method to Estimate the Timestamp Accuracy of Measurement Hardware and Software Tools. In: Proceedings of Passive and Active Measurement Workshop (2007)
29. Zseby, T., Molina, M., Duffield, N., Niccolini, S., Raspall, F.: Sampling and Filtering Techniques for IP Packet Selection, http://www.ietf.org/internet-drafts/draft-ietf-psamp-sample-tech-07.txt