Accuracy evaluation of application-level performance measurements

Katarzyna Wac$^{1,3}$, Patrik Arlos$^2$, Stefan Chevul$^2$, Lennart Isaksson$^2$, Markus Fiedler$^2$, and Richard Bults$^3$

$^1$ University of Geneva – Center for Informatics, 1211 Geneva, Switzerland
   katarzyna.wac@cui.unige.ch
$^2$ Blekinge Institute of Technology – Telecommunication Systems Dept.
   371 79 Karlskrona, Sweden
   \{patrik.arlos, stefan.chevul, lennart.isaksson, markus.fiedler\}@bth.se
$^3$ University of Twente – Computer Science Dept., 7500AE Enschede, the Netherlands
   \{wac, bults\}@cs.utwente.nl

Abstract. In many cases, application-level measurements can be the only way for an application to evaluate and adapt to the performance offered by the underlying networks. Applications perceive heterogeneous networking environments spanning over multiple administrative domains as “black boxes” being inaccessible for lower-level measurement instrumentation. However, application-level measurements can be inaccurate and differ significantly from the lower-level ones, amongst others due to the influence of the protocol stacks. In this paper we quantify and discuss such differences using the Distributed Passive Measurement Infrastructure (DPMI), with Measurement Points (MPs) instrumented with DAG 3.5E cards for reference link-level measurements. We shed light on various impacts on timestamp accuracy of application-level measurements. Moreover, we quantify the accuracy of generating traffic with constant inter-packet-times (IPTs). The latter is essential for an accurate emulation of application-level streaming traffic and thus for obtaining realistic end-to-end performance measurements.

1 Introduction

Application-level measurements are, in most cases, the only way for an application to evaluate the performance offered by the underlying heterogeneous networking environment spanning over multiple administrative domains. In this context it is hard, for not saying impossible, to insert probes along the application’s end-to-end communication path. An application sees the network as a “black-box” transport system accessible via TCP or UDP, and may use measurements to adapt to the perceived network conditions. A complication rises
from the fact that the observed application-level behavior can be different from
the behavior observed at the link layer due to the influence of, for example the
sender and receiver hosts’ protocol stacks on the packets’ generation and ac-
quision processes, the Operating System (OS), the system clock influencing
application timestamps accuracy, or even the application itself. However, if a
host is not overloaded and the protocol stack is properly implemented, certain
parameters, e.g. inter-packet-time for a certain load, ideally should be the same
at both: application layer and link layer. With this in-mind we evaluate the ac-
curacy of application-level measurements with comparison to the reference
link layer measurements.

We study two different application-level active-measurements-based tools; a
tool developed in the MobiHealth project [1] (referred further as the tool A) and
in the Personal Information in Intelligent Transport systems through Seamless
communications and Autonomous decisions (PIITSA) project [2, 3] (referred fur-
ther as the tool B). The tools are intended to evaluate: speed$^4$ (tools A and B)
and dependability$^5$ (only the tool B) related performance criterion for an user
information transfer function (along the ITU-T recommendations I.350 [4], I.380
and the newest Y.1540 and Y.1541 [5]). However, since the only value the tools
can dynamically measure$^6$ is time, they use this parameter (via the obtained
timestamps) to calculate other parameters. Particularly, the tool A calculates
a packet-level end-to-end one-way-transit-time (OWTT) (i.e., application-level
Protocol Data Unit (PDU) delay) and its variation, and the tool B calculates an
observation-window-based application-level throughput (at sender and receiver
separately), and its variation as well as data loss ratio.

The major differences between the evaluated tools are that the tool A uses
TCP, while B uses UDP for transport, A uses Java for implementation, while B
uses C#, and they use different sleep functions (for generating PDUs at a steady
rate) and PDU timestamp methods (explained further in Section 3). These differ-
ences influence the accuracy of the generated inter-packet-times (IPTs)$^7$ and the
timestamps’ accuracy (which in turn influence the OWTT calculations). We eval-
uate the tools using the Distributed Passive Measurement Infrastructure (DPMI)
[7], with Measurement Points (MPs)s equipped with DAG 3.5E [8] cards.

There are many specialized application-level active measurement tools, how-
ever, as indicated by Michaut et al. [9], there are no guidelines for measurements’
quality assurance and in most cases, tools are simply not evaluated. Most of the
tools are intended to operate under Unix/Linux OS, hence their authors assume
µs resolution for timestamps [10]. Some of the authors theoretically calculate
influence of timestamps accuracy and hosts’ synchronization method on the re-
results obtained with their tools [11]. For example, Pathchar authors indicate [12]

---

$^4$ Time interval used to transport data from a source to a destination [4].

$^5$ The degree of certainty with which the service can be used regardless of speed or
accuracy (i.e., its degree of correctness) [4].

$^6$ Association of numbers with physical quantities and natural phenomena by compar-
ing an unknown quantity with a known quantity of the same kind [6].

$^7$ IPT is an inter-departure-time at a sender or an inter-arrival-time at a receiver.
that their tool uses network delays estimations (with 5% accuracy) to calculate bandwidth, and these are not the main source of tools’ measurement errors. Similarly Ali et al. [13], based on differential calculus theoretically tackled estimation of error introduced by events timing inaccuracies in end-to-end bandwidth measurements by four different tools: Pathload, pathChirp, spruce and IGI. Spruce estimates bandwidth with 23% error, while IGI with 29% error for timestamp accuracy of the order of 10 µs. Moreover, Ali et al. estimate $1 - 6\mu s$ delay for accessing the system clock with gettimeofday() function. They also estimate protocol stack delay in the order of $5 - 65\mu s$ per packet.

The most practical (i.e. measurements-based) evaluation of the selected application-level active measurements tools like ping and J-OWAMP [14] has been already conducted [15], indicating possible measurements inaccuracies if tools operate under Windows XP OS. We extend this study over the other application-level tools.

This paper is structured as follows. In Section 2 we present experimental setup for tools’ evaluation, while in Section 3 we present the tools themselves. In Section 4 we explain analysis method for measurements data and in Section 5 we present measurements results. Finally, Section 6 concludes on our findings.

2 Setup

Figure 1 presents the physical setup used in the experiments. The evaluated tools A and B were installed on the source/destination hosts: H1 and H2. These hosts were identical with respect to hardware (Dell Optiplex), with 667 MHz Pentium-3 CPU, 256 MB RAM and built-in 100 Mbit full-duplex Ethernet cards. The OS was Microsoft Windows XP with service pack 2 (updated from Microsoft on the 26th of June 2006). The choice of OS is motivated by the fact that in a mobile application scenario, the mobile node is likely to be a Windows OS-based mobile device - user’s extension of (mostly Windows OS-based) desktop.

The Java version used was 1.5.0. Both H1 and H2 ran Tardis v1.6 software [16] to synchronize their clocks to a local Network Time Protocol (NTP) server (time.bth.se). Access to this server was obtained via the TS, acting as a router for H1 and H2 for the NTP server access as well as a traffic shaper on the traffic sent between the H1 and H2 (thus not influencing the routed NTP traffic).

The link between H1 and TS, as between H2 and TS, passed via wiretaps [17], which tapped the traffic and sent it to two MP: MP03 and MP12, that are a parts of a DPMI setup [7]. Both MPs were equipped with Endace DAG 3.5E cards [8], synchronized using a TDS-2 connected to an Acutime Global Positioning System (GPS) antenna, yielding a timestamp accuracy of 60 ns [15]. The wiring from the wiretaps to the MPs were done such that one MP monitored the traffic in one direction on both links, i.e., MP03 captured traffic going from H1 to TS and from TS to H2, while MP12 captured traffic from H2 to TS, and TS to H1. This way, the link layer PDU’s timestamps are obtained from the same source, i.e., DAG card at MP’s, and (locally) synchronized.
The MPs were in turn connected to a dedicated Gigabit Ethernet (GE) switch, to which a Consumer and a Measurement Area Controller (MArC) also were connected. The MArC managed the MPs, while the Consumer analyzed (in real-time) the streamed measurement trace from the MPs, without influencing the measurement process and its accuracy. The consumer ran also the analysis tools (detectors of setup problems, analyzers of measurements infrastructure’s parameters like IPT and link utilization, catchers of data discrepancies), as well as a data dumper, storing the measurement data into files.

The evaluated tools A and B were configured to generate over the period of 25 minutes a load on the network, comparable to a healthcare application load while sending eight channels of patient’s vital signs data, equivalent of a user-payload size of 526 or 538 B (TCP/UDP) (corresponding to one link-level PDU of 576 B), with a nominal IPT of 125 ms and 90 ms. For more information about the content of the load and see [1].

\section{Tools}

Both tools have a configurable load generator (in terms of PDU length, IPTs and number of PDUs) emulating a real application-level load at the ingress points of the underlying heterogeneous networking environment. Both tools have also dedicated measurement functions for acquiring and correlating measurements (PDU timestamps) at the ingress and egress points of the underlying networking environment, and for generating statistically analyzed measurements results in a textual and graphical form (offline). In both tools, during the measurement session all the data is saved into a static vector, containing of columns like: PDU sequence number, its rate/IPT, length and timestamps, saved to a text file after a session. Moreover, the tools are cooperative [9], i.e., consist of separate sender and receiver program respectively installed on the sender and receiver hosts.
3.1 Tool A

The tool A has been developed and used in the European Union 6th Framework MobiHealth project to evaluate the end-to-end performance in a heterogeneous networking environment (with WLAN/GPRS/UMTS as wireless technologies) supporting time-critical mobile healthcare services [18]. The tool calculates the OWTT on a per-message basis and its variation. Based on the OWTT, an application-level throughput can be derived (all measures are derived offline). The critical requirement for this tool is to have sender and receiver’s time clocks precisely synchronized (by e.g. NTP) in order to get usable timestamps.

The tool was implemented in Java v.1.3.0, to comply with e.g. IBM J9 JVM used on mobile devices. The tool has a simple Text User Interface disabled when the measurement session is running. TCP is used as the transport system interface, with explicit data flush after PDU send to the socket. It uses JavaThread.sleep() functions to send PDUs at a given IPT. It is important to note that the tool, based on the given IPT, calculates and then tries to keep the required number of PDUs per time-window of 1 second. That implies change of the IPT of the last PDU send in the window, such that the sum of all IPTs in the window equals to 1 s. The receiver is implemented such that it continuously attempts to receive data from the socket. The tool uses System.currentTimeMillis() to obtain a PDU timestamp just after each sent or received PDU. For each measurement session, the Java thread had the priority set to HIGH in the OS.

3.2 Tool B

Tool B has been developed and applied in the Swedish PIITSA project to achieve goals as for tool A, but in context of multimedia services [2, 3]. The tool provides input for calculations of an observation-window-based application-level throughput statistics (at sender and receiver separately) reflecting variations and data losses (derived offline). For this tool, there is no requirement for the nodes’ clocks to be time-synchronized.

Figure 2 presents the load generation and measurement function workflow at the sender side. Each sent PDU contains its sequence number inside (for loss ratio detection). At the start of PDUs stream generation, a reference timestamp (T0) is acquired and used further as an absolute value to all upcoming timestamps. In a PDUs stream, the IPT function (an active waiting while loop), based on timestamps T0, T2a and T2b, tries to keep the required IPT value. If some IPT is not fulfilled, e.g. due to send function delays, the next IPT will be shortened to correct this as precisely as possible, without influencing the upcoming IPTs. The sender take timestamps (T3,T4) before and after each PDU send function. The receiver is implemented such that it continuously listens for data to be received from the socket, and it timestamps each PDU upon reception.

The tool has been developed in C#, using .NET framework. To achieve a µs timestamp resolution, the tool uses performance counters, especially the kernel32.dll QueryPerformanceCounter and QueryPerformanceFrequency functions, in conjunction with the system time. Timestamps are calculated by
dividing the counter value by the frequency value. The evaluation of the CPU frequency is only done during the measurements initialization, and its assumed constant during the measurements session. The tool has a simple Graphical User Interface disabled when the measurement session is running. For each measurement session, the tool’s priority has been set to \textit{realtime} in the OS.

4 Measurement data analysis

The experiments were done by having host H1 and H2 running the tools (one at a time) and collecting the application-level traces, while DPMI collecting the link layer traces. The measurement data was then analyzed offline using Matlab 7. From the timestamps, the IPTs was calculated at the application- and link-layer at both the sender and receiver. Moreover, the timestamps’ accuracy error, $T_{\Delta}$ as observed at the receiver’s application-layer over the whole measurements session has been calculated, assuming the link layer IPT values as reference values. The $T_{\Delta}$ value comprises of all possible extreme inaccuracies as it combines all error sources that can affect a timestamp, e.g. errors due clock resolution, due to the clock skew, due to the clock drift, due to clock access time or synchronization errors/events\cite{15}.

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 $IPT_{x,y}(k, k+1)$ be an IPT for a PDU pair $(k, k+1)$ and $\epsilon_{k,k+1}$ a timestamp accuracy error for this pair, then $T_{\Delta}$ is obtained using:

$$IPT_{x,y}(k, k+1) = T_{x,y}(k+1) - T_{x,y}(k)$$
$$\epsilon_{k,k+1} = IPT_{a,r}(k, k+1) - IPT_{l,r}(k, k+1)$$
$$T_{\Delta} = |max(\epsilon_{k,k+1})| + |min(\epsilon_{k,k+1})|$$

5 Results

In the following subsections we present the IPTs at the application- and link-layer as a function of PDU sequence number at the sender and receiver for both tools A (Fig. 3) and B (Fig. 4) under the load of nominal IPT of 125 ms.

Fig. 2. Tool B load generator and measurement function at sender.
In the following tables and we summarize statistical characteristics of the IPT measurements under the load of nominal IPT of 125 ms (Tab. 1) and 90 ms (Tab. 2) over the whole measurement sessions, emphasizing the $T_\Delta$.

![Graphs showing IPT measurements for Tool A at sender and receiver for nominal IPT 125 ms](image)

**Fig. 3.** Tool A: measured IPT at sender and receiver for nominal IPT 125 ms

<table>
<thead>
<tr>
<th>Tool</th>
<th>S-App</th>
<th>S-DL</th>
<th>R-DL</th>
<th>R-App</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tool A</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>120.00</td>
<td>109.88</td>
<td>109.96</td>
<td>20.00</td>
</tr>
<tr>
<td>max</td>
<td>231.00</td>
<td>236.76</td>
<td>236.92</td>
<td>241.00</td>
</tr>
<tr>
<td>mean</td>
<td>125.43</td>
<td>125.43</td>
<td>125.43</td>
<td>125.43</td>
</tr>
<tr>
<td>median</td>
<td>130.00</td>
<td>125.41</td>
<td>124.96</td>
<td>130.00</td>
</tr>
<tr>
<td>std.dev</td>
<td>5.11</td>
<td>1.13</td>
<td>1.23</td>
<td>5.33</td>
</tr>
<tr>
<td>$T_\Delta$[ms]</td>
<td>N/A</td>
<td>209.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>S-App</th>
<th>S-DL</th>
<th>R-DL</th>
<th>R-App</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tool B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>65.86</td>
<td>65.97</td>
<td>65.98</td>
<td>65.97</td>
</tr>
<tr>
<td>max</td>
<td>184.78</td>
<td>184.86</td>
<td>184.94</td>
<td>184.94</td>
</tr>
<tr>
<td>mean</td>
<td>124.91</td>
<td>125.00</td>
<td>125.00</td>
<td>125.00</td>
</tr>
<tr>
<td>median</td>
<td>124.93</td>
<td>125.00</td>
<td>124.96</td>
<td>124.95</td>
</tr>
<tr>
<td>std.dev</td>
<td>0.86</td>
<td>0.81</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>$T_\Delta$[ms]</td>
<td>N/A</td>
<td>3.45</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 1.** Tools A, B: IPT’s statistics at sender and receiver for nominal IPT 125 ms

From the results we conclude that the tool A, using Java thread sleeping functions to generate PDUs at a given IPT, indeed generates PDUs at the required

---

8 S-App and S-DL is a sender's application and link level behavior, correspondingly R-App and R-DL is the receiver’s behavior.
IPT of 125 ms (as we see from the reference link layer data, left side bottom) but the application-level measurements, based on System.currentTimeMillis() timestamping method are inaccurate, mainly due to method’s resolution of 10 ms under the Windows OS (left side top). For the nominal IPT of 90 ms, the IPT inaccuracy at the application-layer originates not only from the timestamps resolution, but also from the PDU generation method at the sender side, where for each 11th packet in the time-window of 1 second the load generator changes the IPT value to 100 ms, such that the sum of all IPTs in the window equals to 1 s. Inaccuracies resulting from this PDU generation method cause high IPT jitter at the sender side, which propagates then to the receiver side.
Comparing application-level with link-layer results for both nominal IPTs, we conclude that for the tool A IPTs inaccuracies are equally distributed in the sender and receiver protocol stacks, resulting from e.g. unpredictable Java thread scheduling in the OS. For both nominal IPTs, the receiver perceives the IPT at the application-layer with a comparable, but slightly higher variance, than the IPT variance at the application-level sender’s side. However, even if mass of timestamp accuracy errors is of the order of 10 ms, an extreme IPT accuracy error $T_\Delta$ at the receiver is large and equals to 209.00 ms for the nominal IPT of 125 ms and 100.50 ms for the nominal IPT of 90 ms, which is more than the required IPT value itself. Despite all the inaccuracies, the throughput statistics derived by this tool can still serve as a rough application-layer estimation. Another observation is the deterministic characteristics observed after the traffic shaper at the link layer (right side bottom).

The C#-based application tool B, using active waiting while loop for PDU generation and performance counters in conjunction with the system time for PDU timestamping, generates PDUs at the required IPT of 125 ms or 90 ms (as we see from the reference link layer data). For the nominal IPT of 125 ms the influence of the sender and receiver protocol stacks on the IPT is minimal. The receiver perceives the IPT at the application-layer with the same variance, as the IPT variance at the application-level sender’s side. However, for the nominal IPT of 90 ms, we see the influence of the receiver’s protocol stack, increasing the IPT jitter almost twice comparing to the sender. For this measurement we have not set the tool’s priority to realtime in the OS at the receiver side (it has been set at the sender), which could influence the resources scheduling for the tool in the OS, therefore resulting in higher inaccuracies. Still the IPT accuracy error shows shows low value, $T_\Delta$ of 3.45 ms for the nominal IPT of 125 ms and error of 15.37 ms for the nominal IPT of 90 ms, indicating minimal influence of the receiver’s protocol stacks on IPT. The throughput statistics derived by this tool are accurate application-layer estimations. Also in this case a deterministic characteristics is shown after the traffic shaper at the link layer (right side bottom).

6 Conclusions

In this paper we have analyzed and evaluated the accuracy of the application-level active-measurements using passive link-layer measurements. We have observed that the application-level behavior can be different from the behavior observed at the link layer due to the influence of the OS at the sender and receiver as well as their hosts’ protocol stacks on the packets’ generation and acquisition processes (especially the Java-based tool A) and the system clock resolution under the Windows OS influencing application timestamps accuracy.

The main conclusion on our findings is that we should always (passively) evaluate the accuracy of an application-level measurement tool before relying upon the results obtained with the tool by, for example, adapting the application’s behavior to the observed underlying network’s behavior. In critical cases,
measurement errors and observation discrepancies may result in the application adaptation resulting in its crash (e.g. buffer overflows) which should be avoided at all cost, especially in time-critical mobile services like for example these in the healthcare domain.

References