Characterisation of the Effect of a Measurement Library on the Performance of Instrumented Tools

Olivier Mehani*,[1], Guillaume Jourjon[1], Jolyon White[1], Thierry Rakotoarivelo[1], Roksana Boreli[1], Thierry Ernst[2]

*Corresponding author: olivier.mehani@nicta.com.au
[1]Nicta, Sydney. Eveleigh, NSW, Australia, first.last@nicta.com.au
[2]Inria, Imara Team, Paris, Rocquencourt, France, thierry.ernst@inria.fr

Published: May 16, 2011

Abstract

The OMF Measurement Library (OML) is an instrumentation system which enables an experimenter to process any type of measurements from distributed applications and collect them in a unified way. We present a comprehensive study of the performance of this library. The analysis focuses on the behaviour, accuracy and precision of instrumented applications, as well as the background footprint such as CPU load and memory usage. To this end, we have modified typical network measurement tools (Iperf) and libraries (libtrace, libsigar) to use OML as their reporting channel. Following extensive experiments, we find little or no negative impact of OML when comparing the OML-enhanced tools to their original versions. Moreover, in the case of Iperf, when we find significant differences, they are positive, with improvements in the accuracy of both the network probing and jitter measurements. We discuss the implications of using OML in the context of experiment-based networking research and give recommendations on its use and the analysis of the produced results.

1 Introduction

Measurement is a foundation stone of scientific research. Many types of measurements must be made using some device or tool, which will have limited accuracy and precision, and may even influence the phenomena that the scientist is attempting to observe. Therefore, without characterising the tools, it is impossible to assess the validity of conclusions that are based on these measurements.

Nevertheless, in the networking community, many published experimental studies shed little light on the tools and methods they use to produce their measurements. This often makes it difficult to accurately reproduce experimental results found in the networking research literature, as compared to some other "hard sciences." As an example, [10] surveyed all papers accepted at a renowned conference of the networking field of research and found that, in many cases, their evaluation sections could be significantly improved with respect to established scientific criteria. The major areas identified for improvement were in the reporting of measurement precisions and the description of the experimental methods used.

Networks are closely linked to the software that embodies and controls them. Many of the tools used to observe network behaviours consist of software executing alongside the processes under study, on the same machine and operating system; if not designed carefully such tools risk disrupting the performance, or even the functionality, of the systems they measure. It should therefore be of great interest to researchers to characterise the influence of software measurement tools on their experiments, and to assess the accuracy and precision of the measurements they produce. Curiously, although there are numerous published software tools for network-based measurement, few of them have been thoroughly studied in this way.

Among the best known Internet measurement tools are the venerable *tcpdump* and *Iperf*. Tcpdump, a command line tool for packet capture, has been shown to accurately report at capture rates up to gigabits per second [19]. Iperf allows researchers to generate a traffic load to evaluate the capacity of a network or resilience of a system; the authors of [11] showed that it generated the highest load on networks paths compared to a number of other traffic generators.

Tools like Iperf and tcpdump are stand-alone and generate measurements in application-specific formats that are stored locally. This limits their usability in large-scale distributed measurements, particularly Internet measurements, where observations from several tools on many network hosts must be correlated. For instance, CPU and memory usage may need to be cross-analysed with application and network behaviour such as throughput and jitter, across many nodes. Therefore, the networking community can benefit from a distributed infrastructure enabling these sorts of flexible measurements. Additional desirable features include the ability to perform real-time remote queries on collected data, support for disconnected modes of operation, and measurement-based steering of running experiments. This kind of architecture has been discussed in depth for some time, in particular in the GENI project through the I&M cluster.[1] Nevertheless, no standard procedure has yet been accepted or evaluated to implement this vision.

Advanced frameworks already exist that address these issues, such as PlanetFlow [6] and CoMon [16] which provide flow logging and slice or node monitoring for PlanetLab, including sophisticated query mechanisms; CoMo [8] is a similar distributed flow measurement tool but not tied to PlanetLab. MINER[2] and OML [23], on the other hand, specialise in instrumenting applications and shepherding their measurements into central databases for convenient real-time or offline analysis, without being tied to a specific type of measurement or platform. There are no studies that characterise these measurement collection tools and platforms in terms of their effects on the accuracy and precision of the measurements of the underlying systems they are helping researchers to observe.

Our contribution in this paper is to provide a comprehensive study of one measurement framework, OML, in terms of the potential biases it introduces in measurements it collects, and its impact on the performance of tools that use it for instrumentation. OML is described in detail in Section 2.1; we used version 2.5.0, the most current at the time. We chose to evaluate the popular tools Iperf (version 2.0.5) and libtrace [1] (version 3), and to this end instrumented them using OML; what they measure and our approach to instrumenting them are discussed in Section 2.2. We then designed a series of experiments (Section 2.3) to determine what effect OML instrumentation has on application performance and measurement accuracy of our tools. We investigated the impact of several experimental factors on various dependent variables for both application performance (Iperf) and accuracy of measurement (libtrace), using analysis of variance (ANOVA) techniques, for which the results are presented in Section 3. This allows us to quantify the operating ranges and scenarios where the effects of OML are significant and where they are negligible. We discuss this at length in Section 4; the overall finding is that when statistically significant differences are found the OML-instrumented tools usually provide better or equal performance.

Our results support the use of OML as a tool for network researchers due to its simplicity of use. Even a naive instrumentation implementation using OML can perform equivalently to a sophisticated hand-coded measurement collection strategy if the measurement rate is not too high, and we provide

[1]http://groups.geni.net/geni/wiki/GeniInstMeas
[2]http://miner.salzburgresearch.at/

recommendations to researchers on how to achieve this in Section 4.4. This study itself also shows the advantages of OML in a distributed environment as it would not have been possible to perform the volume of experiments presented here if we did not have OML to provide marshalling of results with coherent timestamping from a variety of sources into one convenient central database.

2 Method

In order to evaluate the effects of OML 2.5.0, we have designed several experiments. We compared performance indicators and assessed measurement accuracy between the original and OML-instrumented software tools. This section first presents some background information about OML and our modifications to the Iperf traffic generator and the libtrace packet capture library. We then describe our experiment designs and procedures.

2.1 OML

OML [23] is a multithreaded instrumentation and measurement library, which was first developed as a component of OMF [18], but is now a stand-alone open source software[3] which can collect any type of measurements from any type of distributed applications and store them in a unified format. Measurement reporting via OML can be added alongside original reporting mechanisms or as their replacement. A unified approach using OML to collect measurements allows effortless correlation of data from different distributed sources to investigate network anomalies, or test research hypotheses or developed prototypes.

OML has three components that allow a user to automatically generate and collect measurements. First, a developer defines Measurement Points (MP) within their applications or services. An MP is an abstraction for a tuple of related metrics which are reported ("injected") by the application at the same instant. At run-time the experimenter can request all or a subset of these MPs to generate Measurement Streams (MS). Samples from unselected MPs are discarded, as they are deemed irrelevant for the current experiment. MSs can be instructed to report samples from their MPs as soon as a specified number (one or more) of samples has been injected, or compute an aggregate at a defined frequency. Before being streamed towards repositories to be stored for later analysis, MSs can also be further processed.

This processing is done through OML's filtering mechanism, which extends the periodic aggregation mentionned above. An experimenter can specify that a function be applied on some of the fields of an MS to format the data or compute more specific metrics. For example, for an application reporting the size of each packet it receives in an MP, a filter may be used to sum these samples

[3] http://oml.mytestbed.net

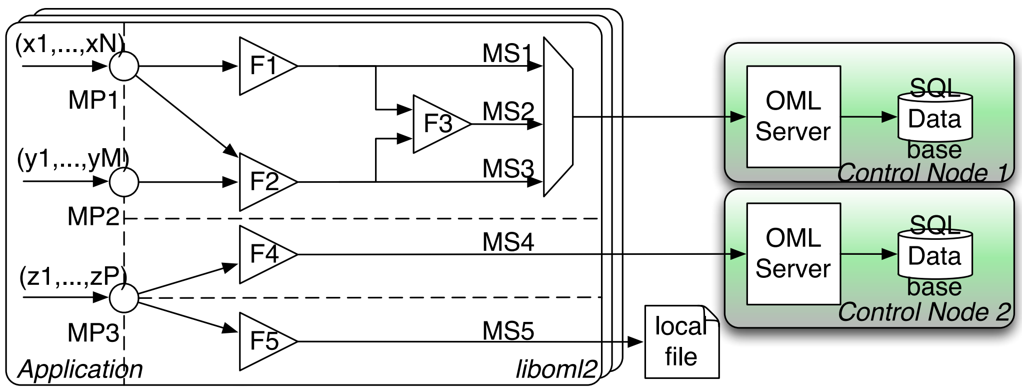

Figure 1: Measurement data path in OML. Three measurement points are filtered to generate five measurement streams (source: [23]).

over a 1 second period to provide an estimate of the immediate throughput. This throughput can then be further processed by an averaging filter with a 1 minute period. OML provides some generic filters such as the aforementionned, but also exposes an API so more specific filters can be easily developped.

Figure 1 shows an example OML data path. An application injects measurement into three MPs. At run-time, the tuples generated by injections in the MPs are combined in order to form five MSs. These newly created streams are then filtered, and the results are directed to one of two different collection servers or a local file. The right part of Figure 1 represents the server side where the OML server serves as a front-end to a database.

By default, MSs are sent reliably to the server using TCP. However, if the network path used for reporting experiences transient losses or cannot provide a sufficient capacity, this may cause the buffers at the OML client to fill up, and samples could be lost before having been sent to the server.

OML also provides a timestamping mechanism based on each reporting node's time (`oml_ts_client`). Each server remaps the MSs they receive to an experiment-wide timebase (`oml_ts_server`) which allows some time comparisons to be made between measurements from different machines. This mechanism however does not remove the need for a good time synchronisation between the involved experimental nodes.

OML has been integrated in many applications, such as traffic generators, passive network measurements, GPS coordinate recorders, and pressure/temperature sensor monitors.[4]

[4]http://oml.mytestbed.net/wiki/omlapp

2.2 Instrumented Tools

For this study, we instrumented tools for network measurement (network probing, packet capture). As a recent study suggests that Iperf is the best performing network probing tool [11], we have chosen this tool as the traffic generator. For network measurements, we use the canonical tcpdump, as well as a more recent packet capture library, libtrace [1]. In addition, we also instrumented a system metrics-measurement library in order to allow us to observe the potential additional load induced by the use of the tools we are characterising. This section details these instrumentations.

2.2.1 Network Probing: Iperf

Iperf is an open source network probing tool.[5] It allows an experimenter to test the characteristics of a network path using either TCP or UDP. Its code is multithreaded to limit the impact of reporting—either on the console or in a CSV file—on the high-speed generation of probe packets. Several patchsets exist to support other transport protocols such as DCCP or UDP-Lite.[6] This tool's versatility and ease of use have allowed it to gain widespread use in the network community, both for traffic generation [7, 12] and as a measurement tool [15, 17, 21, 22].

Iperf can report a number of metrics depending on the transport protocol in use. For TCP only the transferred size, from which the throughput is derived, can be observed. For UDP, packet loss and jitter information can also be reported. The periodicity of Iperf's reports is configurable from once for an entire run to as frequently as every half a second. The internal aggregation function depends on the metric: the transferred size and losses are summed, while the latest value to date is reported for the jitter.

There tends to be some confusion with respect to the definition of what jitter is [3]. In the case of Iperf, the term refers to the variation in packets transit times, as described in [20]. It is computed at packet p as

$$\begin{aligned}\tau_p &= T_p^{Rcv} - T_p^{Snd}\\ \Delta\tau_p &= |\tau_p - \tau_{p-1}|\\ J_p &= J_{p-1} + \frac{1}{16}(\Delta\tau_p - J_{p-1}).\end{aligned} \tag{1}$$

As this jitter itself is based on the *variation* of transit times, rather than the immediate values, it is rather robust to loose time synchronisation between sender and receiver.

[5] http://iperf.sourceforge.net/

[6] http://www.linuxfoundation.org/collaborate/workgroups/networking/Iperf

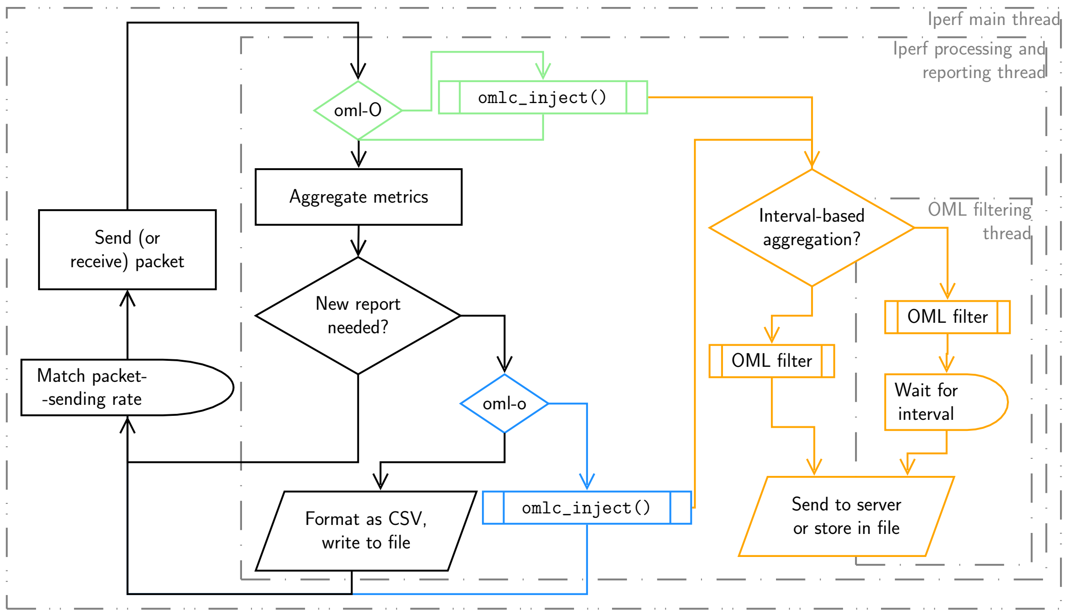

Figure 2: Iperf main loop (black) and OML instrumentation additions (colours). OML reporting is implemented in two sections. The legacy mode (oml-o, blue) reports aggregate metrics computed by Iperf, while the advanced mode (oml-O, green) reports each packet. In both cases, the data can be filtered before being sent for storage (orange). Iperf can delegate reporting to a second thread if activated, while OML uses a filtering thread for time-based aggregates (*e.g.* per-second averages).

We have instrumented version 2.0.5 of Iperf to support reporting via OML.[7] Two separate modes of operation have been implemented in the form of new reporting styles, *legacy* (`iperf -y o`) and *advanced* (`iperf -y O`), which differ in the amount of processing that is done in the application. Table 1 summarises the performance metrics directly reported by the different flavours of Iperf used in this study. Figure 2 shows the main traffic-generating loop of Iperf, and how the OML instrumentation has been integrated into it.

In the legacy mode, the aggregation of the measurements is done using Iperf's standard code, and the periodic reports are sent out through OML via three MPs: `transfer` for the size, `losses` for lost and sent datagrams, and `jitter` for Iperf's implementation of (1). In the advanced mode, Iperf directly reports information about each packet sent or received via OML, in the `packets` MP which contain identification, size and both sent and received (if relevant) times for each packet, down to the microsecond. The advanced mode is more in line with OML's approach, where the measured data is reported ver-

[7]The latest instrumented code is available at http://omf.mytestbed.net/projects/Iperf/repository/show?rev=oml/master

Table 1: Summary of the information reported by the various flavours of Iperf considered in this study.

transport / *flavour*	**Reliable stream**	**Unreliable datagrams**
Vanilla	Transferred size, Throughput,	*idem* + Losses, Jitter
OML legacy	Transferred size	*idem* + Losses, Jitter
OML advanced	Packet ID and size, emission and reception timestamps	

batim by the application and all processing and consolidation is done through filters, thus allowing more experiment-specific treatment without impacting the main operation of the application. Noting that, in most of the literature based on Iperf, there is a lack of precise reporting of versions, platforms and parameters, we also implemented MPs to report such ancillary information about the experiment as the version numbers and command line arguments.

The code we modified to integrate OML is part of the main sending/reporting loop of Iperf. It therefore has a potential to impact the application's performance. Section 2.3.1 describes the experiment we designed to evaluate this effect.

2.2.2 Packet Capture: tcpdump and libtrace

Packet capture in networking environments is usually done via wrapper libraries hiding the operating system's underlying API. Perhaps the most common library for this purpose is the libpcap, derived from work on tcpdump.[8] A more recent library for the same purpose is the libtrace [1], which offers a broader range of input and output APIs and formats.

In this study we focus on the use of both the libpcap in its most simple way with the tcpdump tool and our implementation of a packet-capturing application with OML reporting written as a wrapper around libtrace functions, *oml2_trace*. In our case, both libraries use the Linux Socket Filter [9] to get packets from the kernel.

The main difference between these tools concerns the range of captured frame information. Indeed, in the case of tcpdump, the binary Ethernet frame is dumped in its entirety (up to the maximum `snarflen`, as specified by the experimenter). With oml2_trace, the information is read from the protocol headers and injected into different MPs, thereby giving the experimenter more

[8] http://tcpdump.org

Table 2: Comparison the way information is reported by both packet-capturing applications.

Protocol	**Fields**	**tcpdump**	**oml2_trace**
Ethernet (Radiotap)	Packet ID, MAC addresses, etc. Wireless channel characteristics	Binary dump (possibly truncated so `snarflen`)	`radiotap` MP
IP	ID, length, addresses, etc.		`ip` MP
TCP/UDP	ID, ports, length, etc.		`tcp` or `udp` MPs
Timestamp			All MPs

control on what is collected. Both applications also support Radiotap pseudo headers to provide information about wireless channels.[9]

Though OML provides timestamping on its own, packets captured from the Linux Packet Filter carry precise timing information. As version 2.5.0 of OML does not provide a mechanism for the application to set the `oml_ts_client` of the reports, the capture timestamps are included as fields of the measurement points.

Table 2 summarises how the considered tools make these metrics available to the experimenter. It highlights the much finer granularity available from oml2_trace.

2.2.3 Resource Usage: Sigar

Sigar (System Information Gatherer And Reporter) is a library which provides cross-platform system performance metrics.[10] An OML-instrumented tool, *oml2_nmetrics*, has been developed based on this library in order to enable basic system monitoring.

This tool can report system information such as CPU load, memory used and network operation as well as per-process state and system usage. In our study, we only use this tool to observe the changes in system load following the introduction of OML-instrumented applications.

[9] http://www.radiotap.org
[10] http://support.hyperic.com/display/SIGAR/Home

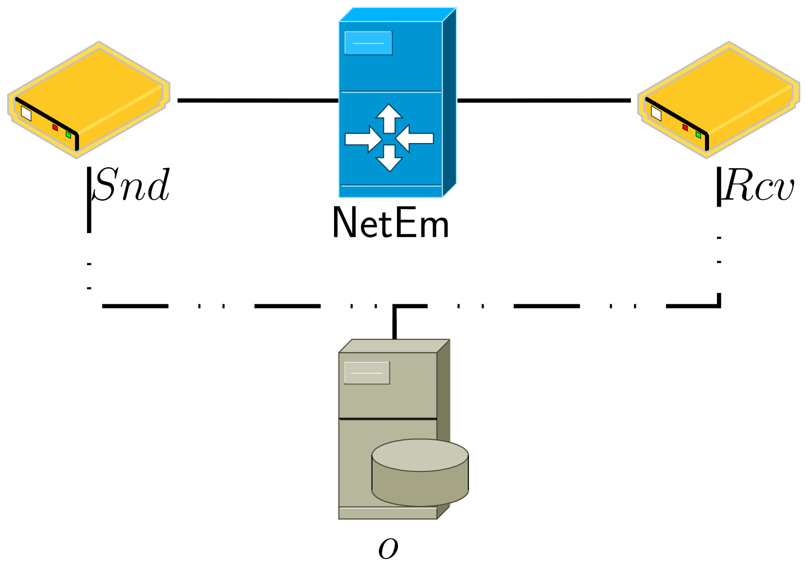

Figure 3: Experiment topology. An Iperf sender Snd generates traffic towards a receiver Rcv on a link capacity-shaped using NetEm. When OML applications are used, measurements are sent over the control network to OML server o.

2.3 Experiments

We use the simple topology illustrated in Figure 3 for numerous experiment trials with varying applications and parameters on the nodes. We used the NetEm emulation software to vary the available link capacity between the sender (Snd) and the receiver (Rcv). We ran two main types of experiments to characterise the OML-instrumented tools and their vanilla equivalents.

To provide a reference for meaningful comparisons, we always run a tcpdump instance on both Snd and Rcv to let us compute the actual rate and jitter *a posteriori*, which we assume to be accurate estimates of the ground truth. We can therefore observe the trials both through the OML-instrumented tools and the tcpdump packet traces, using the following main metrics:

rate R_X^S observed by tool X (i for Iperf, t for tcpdump) at node S (Snd for sender, Rcv for receiver) and,

per-packet timestamps T_X^S from which jitters J_X can be computed at the receiver.

The remainder of this section describes the design of our experiments to evaluate the impact of OML instrumentation on the performance and accuracy of applications, and on the system resources.

Table 3: Experiment design to characterise the impact of the OML instrumentation on Iperf.

nooml-c		**oml-c**	
threads	**nothreads**	**threads**	**nothreads**
598 samples	···	···	598 samples
oml-o		**oml-O**	
threads	**nothreads**	**threads**	**nothreads**
598 samples	···	···	598 samples

2.3.1 Application Performance

The first potential impact we want to test is the effect of OML on an application's performance. In the specific case of Iperf, we are therefore interested in how the traffic generator performance and the accuracy of the reported measurements may be altered. Thus our experiment's null hypothesis is that the OML instrumentation has no significant impact on the packet sending rate of Iperf's traffic generator, and on the accuracy of Iperf's report of measured throughput and jitter.

Experimental Factors Four fixed factors may prove relevant to our working hypothesis above.

Iperf flavour and reporting mode The main concern for this experiment, we want to assess whether the OML instrumentation and the various reporting modes introduce a deviation from the standard version. This could be seen as two nested factors, *i.e.* OML instrumentation or not, with the various reporting modes nested within. However, it made for a simpler design to flatten them into a single factor with four treatments:

nooml-c No modification, CSV reporting to a file;

oml-c OML instrumentation in the code, but CSV reporting to a file; this doubles as a sanity-check to make sure the instrumentation didn't break any mechanism;

oml-o OML instrumentation with legacy OML reporting;

oml-O OML instrumentation with advanced OML reporting.

Required sending rate At high rates the processing of each packet for measurement reporting may take longer than the inter-packet sending period, thus impending the sending rate. Although the sending rate is continuous, we treat it as a fixed factor (1, 10, 50, 95 and 100 Mbps) as our study does not focus on other rate values due to space constraints.

Threads One of Iperf's features is its use of threads (optional, but enabled by default) to report the measurements out of the main high-speed traffic generation loop. In the context of our study, this could hide performance impact introduced by the instrumentation. We therefore considered this factor with two treatments, **threads** and **nothreads**.

Transport protocol Vanilla Iperf supports both TCP and UDP.

Dependent Variables This experiment focuses on three dependent variables which we measure to characterise the potential influence of the OML instrumentation.

Actual sending rate Iperf's real sending rate as computed from tcpdump traces at the sender, R_t^{Snd}.

Accuracy of the throughput report The difference between the throughput reported by Iperf and the one measured by tcpdump at the receiver. For a given sample, $R_{\text{Diff}}^{Rcv} = |R_t^{Rcv} - R_i^{Snd}|$.

Accuracy of the jitter report Similarly, the difference between the jitter reported by Iperf and the one computed from tcpdump at the receiver, $J_{\text{Diff}} = |J_t - J_i|$, where J_t is computed from tcpdump traces as per (1) at the last packet of the period of each sample J_i.

Experiment Design Our final design is shown in Table 3. In this design, we only use UDP as the transport protocol, as it is unclear what proportion of the effect on the dependent variables would be due to the OML instrumentation or the TCP congestion control mechanisms. Also, we decided not to consider the required sending rate as a direct factor, but rather run separate trials at each rate and consider their grouped results separately. Indeed, as the sending rate is one of our dependent variables, it is obvious that changing its set point will have an impact on this observed variable. Another approach would have been to normalise the measured rates with respect to this set point, but the first experiments showed a clear effect which may have belittled other factors of more interest to us.

In each trial, the measurement applications run for 300 s. For each sampling period of 0.5 s, the measured or computed dependent variables are reported. As we only focus on UDP traffic, there is no transient adaptation period as is usually the case with TCP's slow-start. We however ignore the first and last sampling period of each trial as their boundaries do not match the application's starting/stopping times, thus resulting in incomplete data on the observed variables within that period.

In an earlier pilot study with a vanilla Iperf, we did not find any statistical difference between subsequent trials of the same experiment. We therefore assume that having 598 samples from one trial is equivalent to having one sample from 598 trials, thus ensuring suitable replication.

2.3.2 Packet Capture

To evaluate the effect of OML instrumentation on packet capturing, we use a similar setup as above, but with an experiment design involving only two dependent variables. In this case, our experiment's null hypothesis is that the OML instrumentation has no significant impact on the accuracy of packet capturing in terms of the number of observed packets and their timestamps.

Iperf is used to generate traffic at various rates between the two nodes. On each side, a plain tcpdump is used as a *reference* (t) to collect packet identifiers (from the IP header) and timestamps from the generated traffic. In addition on each side, another packet capture application is used as an *alternative* (a) to measure the same variables. This alternative can either be our OML-instrumented oml2_trace application or another tcpdump instance.

Experimental Factors We consider two fixed factors in this experiment. The first one is the sending rate, with the same treatments as before: **1**, **10**, **50**, **95** and **100** Mbps. The second one is the use of the OML-instrumented *alternative* application, or not. This corresponds to the two treatments **trace** (when an OML-instrumented application is used), and **notrace**.

Dependent Variables This experiment focuses on two dependent variables.

Accuracy of packet report Missing packet reports
may introduce a bias in the observations of a given research study. This may happen *e.g.*, when the control network is saturated with large number of reports, resulting in retransmissions and losses of samples. Thus for each trial, we count the number of packets sent N_{sent} based on the identifier from t, and the number l_a^S of packets not captured by a on side S, as identified by gaps in the IP sequence numbers; we then compute the loss ratio $L_a^S = l_a^S / N_{\text{sent}}$.

Accuracy of timestamp The difference in the reported timestamps between t and a on either node, $T_{\text{Diff}}^S = |T_t^S - T_i^S|$.

Experiment Design The timestamp precision T_{Diff}^S can be estimated for each packets in a trial. We therefore have access to many more samples of that variable than in the previous experiment (*e.g.* of the order of 1×10^7 at 50 Mbps). In contrast, the number of unreported packets, N_{Loss}, is a single aggregate value for each trial. We therefore run 25 trials of this experiment to have sufficient replication and an acceptable number of samples of N_{Loss}.

2.3.3 System Resources

To evaluate the impact of OML instrumentation on system resources, for each of the treatment groups in the above experiments we also run the oml2_nmetrics application to monitor resource usage such as non-idle CPU time C^S and used memory M^S. In this context, our null hypothesis is that the OML instrumentation has no significant impact on the CPU and memory usage of the system running it.

Dependent Variables The following additional two variables are observed for each previously described experiments. We remove the trend displayed by both C^S and M^S by differentiating subsequent samples for each time interval $[T-1, T]$.

Differentiated CPU time $\delta C^S = C^S_T - C^S_{T-1}$ and,

Differentiated memory usage $\delta M^S = M^S_T - M^S_{T-1}$.

Experiment Design The experimental factors in this last experiment are the same as the ones for the previous experiments. We run oml2_nmetrics in each treatment groups. As it is an instrumented tool itself, we assumed it introduces a uniform bias to the system load for each trial. We therefore ensure that no further trend biases our dependent variables by starting oml2_nmetrics first to let it measure the idle system before actually starting the other instrumented tools. This doubles the previously mentioned trial duration. Thus with a sampling period of 5 s, we collect a total of 120 samples per node, but only the last 60 are representative of the impact of the measurement tools.

2.3.4 Technical Details of the Experiments

The experiments described before have been conducted on an OMF-enabled testbed. This allows us to provide systematic experiment descriptions suitable for peer-review and reproducibility.[11]

The experimental nodes *Snd* and *Rcv* are VIA MB770, 1 GHz fan-less CPU with 1 GB of RAM with two 100 Mbps Ethernet cards and two wireless interfaces. The measurement server *o* is a 6-core AMD Phenom II X6 1055T Processor with 12 GB of RAM, and is connected to the experiment nodes on

[11] All experiments and analysis scripts used for this paper, as well as our datasets, are available at `http://norbit.npc.nicta.com.au/portal/projects/omlperf`.

a 1 Gbps internet LAN. The experiment nodes run Ubuntu Linux with kernel 2.6.35-25-generic #44-Ubuntu SMP.

3 Results

This section presents the results of the analyses which we performed on the collected experimental data.

Prior to analysing the data, we first assessed the precision of our measurements by computing the relative standard error (RSE) of all measured variables within a trial for each treatment groups of the previously described experiments. The minimum and maximum RSE in our datasets are 8×10^{-6} % and 0.17 %. Thus our collected set of data has a sufficient *precision* (< 5 %) to be used in further scientific analyses.

We now consider the dependent variables introduced in Section 2.3 and attempt to disprove our null hypotheses by identifying significant variations between each treatment groups in our experimental factors. To this end, we perform analyses of variance (ANOVA) on the dependant variables. We choose the significance level $\alpha = 0.05$ to have 95 % confidence when finding significant differences.[12] We note that some of the assumptions of the ANOVA are not always met by our datasets. We address these problems as follows.

Independence of samples As the samples of each variables for one trial come from a time series, they are clearly not independent, which is confirmed by Turning points tests [14]. We therefore make our data iid by sampling it randomly with replacement as suggested in [13].

Homoskedasticity In some cases, Breusch-Pagan tests show that the variance of our samples differs significantly between treatment groups. Studies have however showed that the ANOVA is robust to deviations from this assumption at the price of a small reduction of the confidence $1 - \alpha$ and an increase of the power of the test β [4, 5]. Moreover, we note that these studies focused on ratio of variances as low as 1:2. Even in our extreme cases, computing the ratio of the variances reveals that the heteroskedasticity is much more modest than the cases studied in [4, 5]. We therefore conclude that our performed ANOVAs gave us valid results even with this caveat on the confidence.

Normality This is the assumption from which our data deviated the most, both in terms of skewness and kurtosis (flatness). We characterised this devi-

[12] If a factor is found to have a significant impact, the probability of it being a false positive (Type I error) is $< 5\%$.

ation with a Shapiro-Wilk test for each treatment group and proceeded with an ANOVA if the deviation was not found to be significant. In case the deviation was significant, we used a non-parametric version of the ANOVA which removes the assumption on the source distribution of the data by creating an empirical null distribution through permutation of the samples throughout treatments [2].

3.1 Application Performance

We performed two-way ANOVAs with interactions for each of the variables R_t^{Snd}, R_{Diff}^{Rcv} and J_{Diff} at each of the studied set rates. However, we judged the results for 100 Mbps invalid as the sender was rate-limited by its local network interface in all treatment groups. Characteristic results for other rates are presented below.

3.1.1 Actual Sending Rate

The results of the ANOVA for Iperf's sending rate as measured by an uninstrumented tcpdump, R_t^{Snd}, at rates 1, 50 and 95 Mbps are shown in Table 4. The results of the analysis for set rate 10 are similar to those for 50.

For rates 10 Mbps and higher, there are statistically significant differences in Iperf's sending rates, which are introduced by changes in both the use of threads and OML instrumentation, as well as the interaction of those two factors. When significant, this interaction has to be studied first, which we do in Figure 4 for case 95 Mbps. This figure shows the so-called graph of means, with each treatment of the oml factor on the x-axis and a connecting line linking means for the same treatment of the thread factor. It shows an interaction between the threads and oml factors. For the threads treatment, the oml factor does not appear to have an impact, while its oml-o treatment affects R_t^{Snd} in the nothreads treatment.

The Tukey Honest Significant Differences test allows us to quantify the deviations observed in Figure 4. We present the relevant results allowing to characterise the previous figure in Table 6. For legibility's sake, we only show the mean differences and the p-values. We however include all the differences between interactions which were found to be significant.

Table 6 further indicates that, contrary to our expectations, the version integrating OML performs *better* than the vanilla version. Indeed, with a set rate of 95 Mbps, an unthreaded OML-instrumented Iperf was able, in the same conditions, to send an average of about 130 kBps ($\simeq$ 1 Mbps) more than the vanilla version reporting to a CSV file, which is the same as the threaded version. We found similar significant effects, though of smaller amplitude, at rates 10 Mbps and above.

Table 4: Two-way PERMANOVAs with interactions on the actual sending rate of Iperf, R_t^{Snd}

	d.f.	SS	MS	F	p	**Signif.**
			1 Mbps			
oml	3	2.00×10^6	6.65×10^5	1.01	0.394	–
threads	1	4.62×10^5	4.62×10^5	0.70	0.418	–
oml:threads	3	4.89×10^5	1.63×10^5	0.25	0.864	–
Residuals	3192	2.11×10^9	6.61×10^5			
Total	3199	2.11×10^9				
			50 Mbps			
oml	3	3.55×10^{11}	1.18×10^{11}	22.70	0.001	⋆⋆⋆
threads	1	2.33×10^{11}	2.34×10^{11}	44.76	0.001	⋆⋆⋆
oml:threads	3	3.55×10^{11}	1.18×10^{11}	22.74	0.001	⋆⋆⋆
Residuals	3192	1.66×10^{13}	5.21×10^9			
Total	3199	1.76×10^{13}				
			95 Mbps			
oml	3	6.29×10^{11}	2.10×10^{11}	8.60	0.001	⋆⋆⋆
threads	1	9.03×10^{11}	9.03×10^{11}	37.03	0.001	⋆⋆⋆
oml:threads	3	3.43×10^{11}	1.14×10^{11}	4.68	0.003	⋆⋆
Residuals	3192	7.79×10^{13}	2.44×10^{10}			
Total	3199	7.98×10^{13}				

Significance level: ⋆ 0.05, ⋆⋆ 0.01, ⋆⋆⋆ 0.001

3.1.2 Accuracy of the Throughput Report

Next, we assess the variations of R_{Diff}^{Rcv} between the treatment groups, as an evaluation of the impact of the instrumentation on the accuracy of Iperf's report. Table 5 presents the corresponding ANOVA results.

At 1 and 50 Mbps, no statistically significant difference seems to be due to OML. At the latter rate, threads are the only source of deviation. At 95 Mbps, both factors, as well as their interaction, are found to have a significant effect. However, there is a discrepancy in the sum of squares for factor oml at this rate.

Further investigation of the data revealed that some packet reports in the advanced mode (oml-O treatment) are lost, thus introducing a bias in report accuracy similar to that mentioned in Section 2.3.2. This leads to underestimating (by 3–4 %) the throughput for that case when computing it *a posteriori* based on these reports. Because of the heavy bias introduced by this treat-

Table 5: Two-way PERMANOVAs with interactions difference R^{Rcv}_{Diff} between the actual received rate and Iperf's report.

	d.f.	SS	MS	F	p	**Signif.**
			1 Mbps			
oml	3	3.66×10^6	1.22×10^6	0.44	0.714	–
threads	1	9.62×10^6	9.62×10^6	3.47	0.061	–
oml:threads	3	3.26×10^5	1.09×10^5	0.04	0.990	–
Residuals	3192	8.85×10^9	2.77×10^6			
Total	3199	8.87×10^9				
			50 Mbps			
oml	3	5.41×10^9	1.80×10^9	1.39	0.230	–
threads	1	1.42×10^{10}	1.42×10^{10}	10.96	0.001	⋆⋆⋆
oml:threads	3	6.05×10^9	2.02×10^9	1.56	0.191	–
Residuals	3192	4.14×10^{12}	1.30×10^9			
Total	3199	4.16×10^{12}				
			95 Mbps			
oml	3	7.81×10^{15}	2.60×10^{15}	83300.57	0.001	⋆⋆⋆
threads	1	3.97×10^{12}	3.97×10^{12}	127.03	0.001	⋆⋆⋆
oml:threads	3	8.09×10^{12}	2.70×10^{12}	86.35	0.001	⋆⋆⋆
Residuals	3192	9.97×10^{13}	3.12×10^{10}			
Total	3199	7.92×10^{15}				

Significance level: ⋆ 0.05, ⋆⋆ 0.01, ⋆⋆⋆ 0.001

ment, particularly on the homoskedasticity of the data, it is impossible to draw any conclusion from the analysis of this case.

To shed some light on the causes of these losses, we first verified that the problem did not arise because of a saturation of the control link. We found that, even at high rates (95 and 100 Mbps), the reporting generated a steady 3–4 Mbps, which is well below the 100 Mbps limit of the control network and the interfaces of the experiment nodes.

We also performed another ANOVA for this case, ignoring treatment oml-O. It showed no statistically significant difference ($p > 0.05$) between the other treatments of the oml factor, including OML legacy reporting. We then ran a modified oml-O trial where a summing filter was used at a 0.5 s interval, thus generating the same amount of reports to the server as in the oml-o treatment. An analysis of the data consisting of the previous set, with the reports from oml-O replaced by those from the newly introduced filtered trial suggests

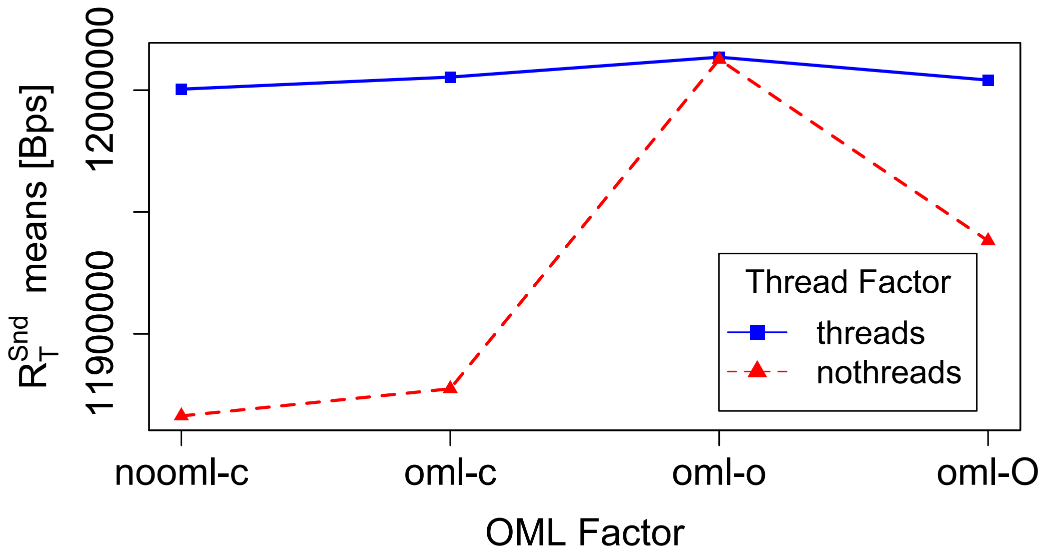

Figure 4: Graph of means for the interaction between experimental factors on R_t^{Snd} at 95 Mbps. Legacy OML reporting allows unthreaded Iperf to achieve a throughput with no significant difference from the threaded version (see Table 6).

that no significant difference can be observed when using interval-based filters. However, as both approaches imply modifications of the experiment design, they cannot be rightly compared to the rest of the results presented here. We keep this as future work for the design of confirmatory experiments.

Based only on the remaining valid results we collected, we found no statistically significant difference in the accuracy of Iperf's throughput which was due to OML instrumentation or reporting. However, although no significant interaction effect was found at 50 Mbps, we still include the graph of means for R_{Diff}^{Rcv} in Figure 5. This graph shows a trend[13] which we find interestingly similar to that of Figure 4, thus suggesting that a similar significant interaction of factors may appear at 95 Mbps.

3.1.3 Accuracy of the Jitter Report

We finally attempt to find differences in J_{Diff} in a similar fashion. Due to the large amount of memory required by our implementation of the *a posteriori* jitter computation based on packet dumps, we could only study the cases in

[13]We do not refer to the apparent "trend" of the lines, which is a meaningless artifact of the graphical representation of categorical data. Rather, we mean the similarity of the behaviours observed for the same combination of factors.

Table 6: Tukey Honest Significant Differences for oml:threads interactions with significant differences in the sending rate at 95 Mbps. Only differences relevant to Figure 4 and all those found to be significant with 95% confidence are shown.

oml:threads interaction	**diff [kBps]**	p **adj**	**Signif.**
oml-c:nothreads–nooml-c:nothreads	−20.77	0.98	–
oml-o:nothreads–nooml-c:nothreads	107.41	0.00	⋆⋆⋆
nooml-c:threads–nooml-c:nothreads	88.27	0.00	⋆⋆⋆
oml-c:threads–nooml-c:nothreads	91.85	0.00	⋆⋆⋆
oml-o:threads–nooml-c:nothreads	108.71	0.00	⋆⋆⋆
oml-O:threads–nooml-c:nothreads	98.91	0.00	⋆⋆⋆
oml-o:nothreads–oml-c:nothreads	128.17	0.00	⋆⋆⋆
nooml-c:threads–oml-c:nothreads	109.04	0.00	⋆⋆⋆
oml-c:threads–oml-c:nothreads	112.62	0.00	⋆⋆⋆
oml-o:threads–oml-c:nothreads	129.47	0.00	⋆⋆⋆
oml-O:threads–oml-c:nothreads	119.67	0.00	⋆⋆⋆
oml-O:nothreads–oml-o:nothreads	−75.13	0.02	⋆
oml-o:threads–oml-o:nothreads	1.30	1.00	–
oml-o:threads–oml-O:nothreads	76.43	0.01	⋆⋆
oml-O:threads–oml-O:nothreads	66.64	0.05	⋆

Significance level: ⋆ 0.05, ⋆⋆ 0.01, ⋆⋆⋆ 0.001

1–50 Mbps. Table 7 summarises our findings, and shows a significant difference introduced by OML in the 10 Mbps ($p < .01$) and 50 Mbps ($p < .001$) cases.

We conducted Tukey HSD tests to evaluate this difference. They showed that no statistically significant difference could be observed between vanilla Iperf with CSV output and OML legacy reporting. However, a significant increase in the jitter was observed in the case of the advanced OML reporting at 50 Mbps, as illustrated by Figure 6. For the 10 Mbps case, the graph of means (not reported here) suggests a similar but much smaller effect. This effect size may be too modest to be detected with statistical significance by our current tests. Tests with higher statistical power may be able to detect such effect and will be considered in our future work.

3.2 Packet Capture

We performed a similar analysis on the relevant dependant variables of our packet-capture experiment. Characteristic results are reported thereafter.

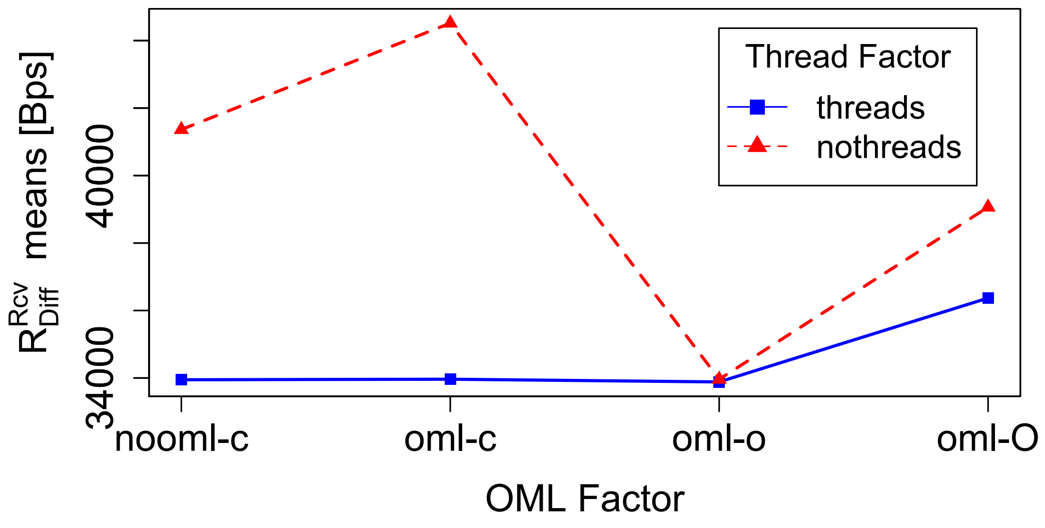

Figure 5: Graph of means for the interactions on Iperf's R_{Diff}^{Rcv} at 50 Mbps. OML reporting appears to increase the accuracy of the reports from the unthreaded trials, bringing them up to that of the threaded ones.

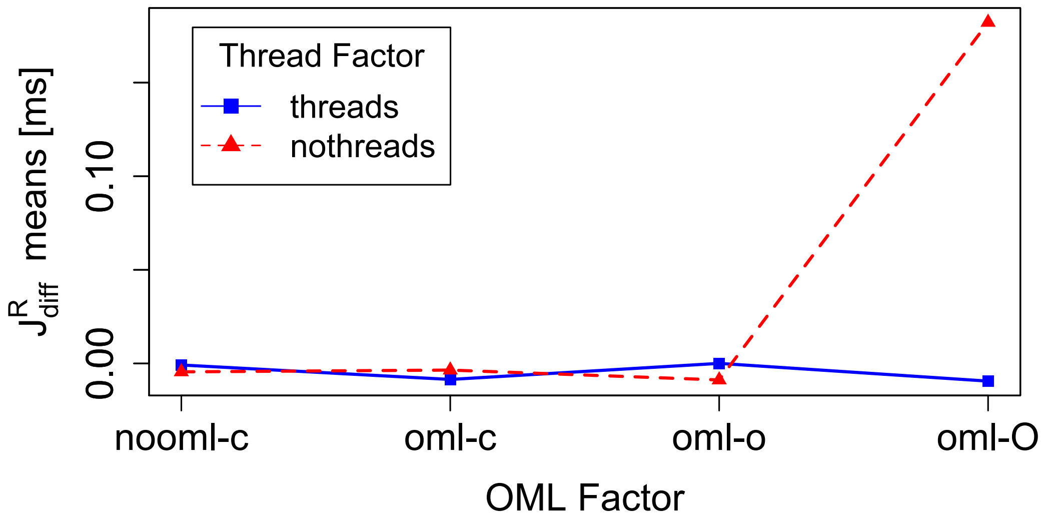

Figure 6: Graph of means for the interactions of experimental factors on J_{Diff} at 50 Mbps. Iperf's legacy OML reporting does not seem to introduce any significant effect.

Table 7: Two-way PERMANOVA with interactions on the difference J_{Diff} between the actual jitter and Iperf's report.

	d.f.	SS	MS	F	p	**Signif.**
			1 Mbps			
oml	3	0.10	0.03	0.48	0.684	–
threads	1	0.00	0.00	0.05	0.833	–
oml:threads	3	0.25	0.08	1.22	0.322	–
Residuals	4784	322.15	0.07			
Total	4791	322.50				
			10 Mbps			
oml	3	0.65	0.22	4.24	0.006	⋆⋆
threads	1	0.11	0.11	2.08	0.149	–
oml:threads	3	0.36	0.12	2.35	0.075	–
Residuals	4561	233.55	0.05			
Total	4568	234.67				
			50 Mbps			
oml	3	1.85	0.62	51.18	0.001	⋆⋆⋆
threads	1	0.63	0.63	52.63	0.001	⋆⋆⋆
oml:threads	3	2.12	0.71	58.72	0.001	⋆⋆⋆
Residuals	4784	57.67	0.01			
Total	4791	62.28				

Significance level: ⋆ 0.05, ⋆⋆ 0.01, ⋆⋆⋆ 0.001

3.2.1 Accuracy of Reports

We performed a two-way permutation ANOVA on the loss ratio L_a^{Snd} for our treatment groups in the packet capture experiment. Table 8 presents a summary of this analysis. Though the trace factor seems to have the most statistically significant impact, its interaction with the rate factor needs to be studied first. The related graph of means in Figure 7 shows that the interaction of the trace factor and high rates has the largest effects, increasing the packet loss ratio.

In Table 9, we reproduce those results of the Tukey HSD test which are relevant to the interpretation of Figure 7. It shows that, at low rates (1–50 Mbps), the difference is not statistically significant but becomes significant for the 95 Mbps treatment. We however note that, regardless of significance, the difference is always positive, which means that our OML-instrumented libtrace tool tends to miss more packets than tcpdump.

Table 8: Two-way PERMANOVA with interactions on the loss ratio L_{ratio}.

	d.f.	SS	MS	F	p	**Signif.**
trace	1	0.07	0.07	9.24	0.001	$\star\star\star$
rate	4	0.08	0.02	2.56	0.027	$\star$
trace:rate	4	0.08	0.02	2.65	0.021	$\star$
Residuals	200	1.48	0.01			
Total	209	1.70				

Significance level: $\star$ 0.05, $\star\star$ 0.01, $\star\star\star$ 0.001

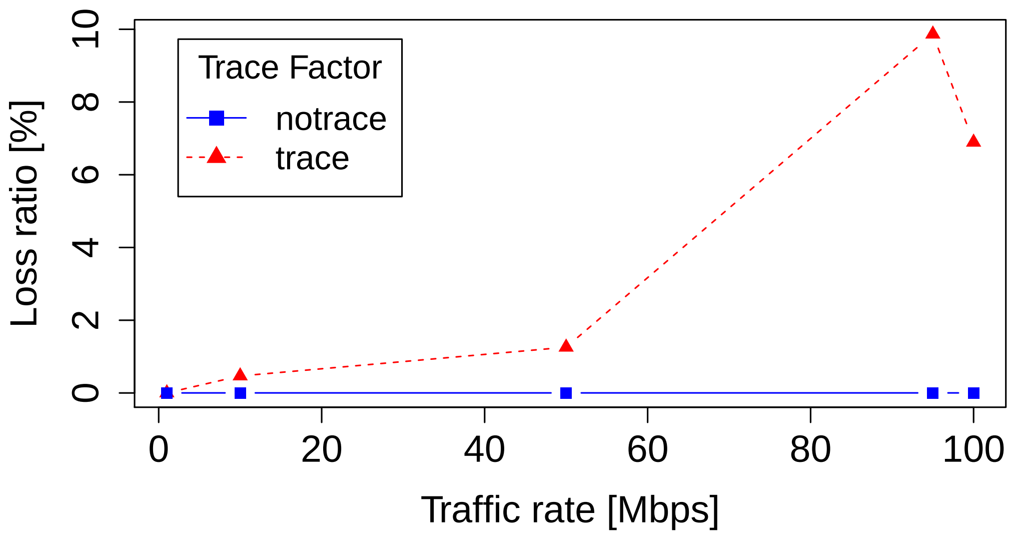

Figure 7: Graph of means for the interaction of factors trace and rate on the loss ratio L_a^{Snd}.

3.2.2 Timestamp Accuracy and Precision

As mentioned in section 2.2.2, the timestamps for both tools come directly from the kernel capture. We found the accuracy for both the reference and the alternative to always be equal ($T_{\text{Diff}}^S = 0$).

It is worthwhile to note that, while both libraries can access and store packet timestamp information with a microsecond resolution, we found in our experiments that the libpcap trace files only had a millisecond resolution. We therefore discarded the microsecond information from oml2_trace when computing T_{Diff}^S.

Table 9: Tukey HSD for trace:rate interactions with significant differences in the loss ratio L_a^{Snd}. Only differences relevant to Figure 7 are shown.

trace:rate	**diff** [%]	p **adj**	**Signif.**
trace:1-notrace:1	9.08e-04	1.00	–
trace:10-notrace:10	0.46	1.00	–
trace:50-notrace:50	1.26	1.00	–
trace:95-notrace:95	9.87	0.02	$\star$
trace:100-notrace:100	6.89	0.26	–

Significance level: $\star$ 0.05, $\star\star$ 0.01, $\star\star\star$ 0.001

3.3 System Resources

We analysed the system usage variables δC^S and δM^S in the same manner as the other dependent variables, for both experiments on Iperf performance and packet capture. Some of the resulting ANOVAs showed significant effects. However, further investigation showed that in these cases the affected node had been doing memory maintenance tasks, thus confounding our results.

This led us to conclude that, overall, no statistically significant deviation could be found, either in terms of CPU or memory usage, between the different treatment groups involving OML-enabled tools or not.

4 Discussion

Based on the interpretation of the results just presented, we now discuss their implications with respect to the performance of OML and the studied applications. We then provide recommendations for the use of OML for the developers and experimenters.

4.1 On the Accuracy of Iperf

Our study of an OML-instrumented Iperf gave us insight, both on the influence of OML and Iperf itself, which we present here.

4.1.1 Vanilla Version

As a side effect of our OML impact study, we have performed what can be considered one of the first deep investigation of Iperf performance and reporting accuracy. This performance analysis of Iperf brought some insight on its potential impact on networking research. In particular, we unexpectedly found that the Iperf reporting function can impact the accuracy and performance of the traffic generation up to 2% in our worst case scenario. This difference can be explained by the reporting functions, which are run synchronously in the non-threaded version of vanilla Iperf.

This discovery raises several questions about previously published Iperf-based measurements. Indeed, although most distributions provide an Iperf built with threads support, some platforms—mostly embedded—do not support threads (*e.g.* Windows CE Pocket PCs).

4.1.2 OML-Instrumented Flavour

Nevertheless, the analysis of the impact of legacy OML reporting on unthreaded Iperf suggests that, even if the reporting is done within the main thread, the use of OML instead of the usual CSV file-writer allows to reach similar performances as the threaded builds. It is important to note that the OML reporting did not introduce any threads in this case.

Our results also show that reporting every packet information for Iperf (advanced mode) could impact the overall performance of both threaded and unthreaded versions of Iperf. Early results from confirmatory experiments however suggest that the use of OML filters before reporting measurements on the network can mitigate this issue. This result is consistent with that of the packet capturing tool as discussed next.

4.2 On Packet Capture With libtrace

Our results for the packet capture software show that there is a significant difference between our OML-instrumented wrapper for libtrace and tcpdump at high rates. In our worst case scenario the OML-based tool would lose an average of about 10 % more packets at 95 Mbps. This behaviour may be explained by two main reasons.

First, the OML architecture may contain a bottleneck which would cause measurements to be lost when generated at too high a rate. Indeed, as illustrated in Figure 8, when a sample is injected by the client, the data stream is transferred reliably to the OML server over TCP. On the server side, the samples are inserted into an SQLite database. The server's default parameter for OML is to perform one writing transaction once every second. In this study, two clients were reporting two MPs (IP and TCP headers) per packet. This resulted in four injections on the server side. At high rates, too many packets may therefore have been reported for the OML server to be able to empty the TCP buffers in time. In those cases, the TCP receiver would start sending zero-window acknowledgements therefore blocking the TCP sender, and finally the injection function. As our basic capturing packet application was not threaded, blocking the measurement injection in turns caused packet capture buffers to be overwritten before being read by the application. The same hypothesis can be proposed to explain the relatively poorer performance

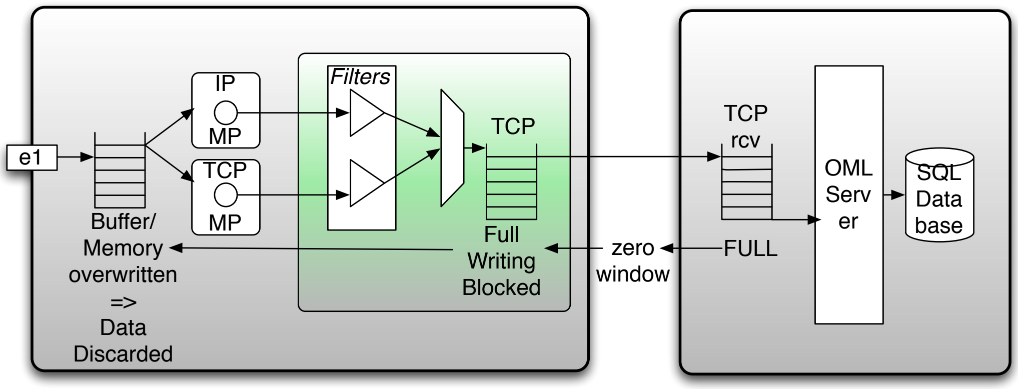

Figure 8: Data path from injection to database. When the transaction takes place on the server side the TCP receiver buffer is filled. As a result a zero window is sent to the client, which results at the other end in data discarding in the application.

of the oml-O treatment of our Iperf experiment.

Also, in oml2_trace we do not dump the binary packet directly into a file as tcpdump does, but rather access specific fields of the relevant headers. This process is done for every packet prior to the injection into the measurement stream. We therefore suspect our application may be slowed down even before the OML process is started. In order to have a fairer comparison, it would be interesting to conduct a complementary experiment. The notrace treatment would either be tcpdump operated in a verbose mode and dumping out the parsed packets in a file, or oml2_trace configured to directly dump the samples from the captured packets to a local storage.

4.3 On the Resource Usage of OML

For all experiments, the OML instrumentation has not been found to induce any significant impact on the system's CPU and memory usage. These results were expected for the experiments at low traffic rates but not for the ones at higher rates. In particular we anticipated that the advanced OML reporting mode of Iperf would show an increased system load, but our results did not identify such impact. Overall, our results suggest that the resource footprint of OML is not significant and can be considered as negligible in most cases.

4.4 Recommendations

4.4.1 OML Instrumentation and Measurements

We claim that OML is a good candidate for the instrumentation of large scale distributed systems. Indeed, aside from its design goal of centralising measurement collection from multiple nodes and storing them in a unified format, our analyses suggest that, in most cases, this library does not have a significant impact on the instrumented applications or the host system. It also simplifies the development of active measurement applications by removing the need for a complex threaded architecture. Nevertheless some considerations have to be taken into account when designing such large scale distributed measurements.

An application developer, be they in charge of writing a new tool or just instrumenting an already existing system, must be careful in establishing which metrics should be reported together in one MP. It is a trade-off between providing flexibility to the experimenter and limiting the number of MPs—and, ultimately, generated samples—that need be reported to the server. For example, it may be relevant to group fields for which new measurements always arrive at the same time (such as various metrics for a packet). However, only some aspects may ultimately be of interest to the experimenter (*e.g.* throughput but not jitter). Thus MPs with a lot of fields may only result in wasted capacity both on the control network and the database server.

Then, an experimenter would have to be careful with respect to the number of MPs selected to be reported to a single OML server, and make sure it does not receive more than a few reports per millisecond. Several solutions exist to do so. First, the experimenter should take advantage of the filtering capabilities of OML to pre-process data on the client side and reduce the number of samples. This solution however requires them to have an *a priori* idea of which metrics may be relevant to their study. It might therefore not be suitable in the exploratory phase of a study.

Another solution is to use an OML proxy server [23]. This should allow to break the lock-up illustrated in Figure 8 as the proxy would decouple the client from the server using its own buffers. This is likely to come at a higher memory price for the node running the proxy.

A third possibility is to multiplex several MSs to dedicated OML servers. For example in the case of oml2_trace this could mean instantiating four OML servers, each corresponding to an MP of a specific node.

4.4.2 Iperf for Capacity Probing

As the results show in this study, it is important to understand all the different components of a measurement application. In particular as illustrated by the case of the non-threaded version of Iperf, it is important to be aware of the impact of the reporting procedure on the measurements themselves.

Based on our findings, we do not recommend the use of unthreaded Iperf builds in studies relying on the absolute accuracy of Iperf measurements. Indeed this lack of thread support may bias the accuracy of the reported throughputs and jitters. However if we assume a uniformly introduced bias, these measurements may be used in studies using relative comparisons, provided that the data was collected from an homogeneous setup of solely unthreaded Iperf builds.

4.5 Limitations and Future Directions

The main limitation found during this study is the potential loss of measurements at high reporting rates. We identified several solutions to mitigate this issue from an experimenter's perspective: instantiating several dedicated servers, preprocessing the samples on the nodes using filters or deploying proxy servers to act as additional buffers.

Based on the aforementioned observations and limitations, several software improvements can be considered to enhance the behaviour of OML. The first one would be to resolve the injection-to-database lock-up problem by separating the measurement injection and the transmission to the server. This could be accomplished by dedicating one thread to the TCP transmission.

Another improvement would be to have feedback updates from the server to the client. They would inform the client of the size of the next database transaction in order to either slow down the sending rate, or change the reporting schema by introducing on-the-fly filters.

Finally, we also plan to investigate filter- and proxy-based solutions further in terms of system impact and achievable sample rate.

5 Conclusion

We have conducted an in-depth performance evaluation of the OML 2.5.0. With a reasonable sample rate, we could not find any significant negative impact of such an instrumentation. Moreover, we identified statistically significant positive effects in some cases where the instrumented application did not use threads. Indeed, OML removes the complexity of reporting from an application's main purpose, therefore making it an interesting and non-intrusive tool to collect metrics from applications which main purpose is not measurement. However, our results indicate that a single OML server cannot currently scale to sample rates of the order of more than a few per millisecond, though preliminary experiments suggest that this issue can be avoided by the use of OML filters.

Incidentally to this evaluation, we could also identify a shortcoming in the performance of Iperf 2. We found that disabling threads on the vanilla version has a significant impact on the accuracy of the generated rate and measurements. This raises concerns about the validity of Iperf-based studies comparing platforms which do not support threads with others which do.

Finally, we presented a set of recommendations for a user to instrument their software with OML and configure the monitoring so this would not impact the performance of the host application. We were also able to narrow down the bottlenecks present in OML, and identify future steps that need to be taken in order to improve the instrumentation library.

References

[1] S. Alcock, P. Lorier, and R. Nelson. Libtrace: A trace capture and processing library. Technical report, University of Waikato, Hamilton, New Zealand, May 2010.

[2] M. J. Anderson. A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, 26(1):32–46, Feb. 2001.

[3] C. Demichelis and P. Chimento. IP packet delay variation metric for IP performance metrics (IPPM). RFC 3393, RFC Editor, Fremont, CA, USA, Nov. 2002.

[4] G. V. Glass, P. D. Peckham, and J. R. Sanders. Consequences of failure to meet assumptions underlying the fixed effects analyses of variance and covariance. *Review of Educational Research*, 42(3), 1972.

[5] M. R. Harwell, E. N. Rubinstein, W. S. Hayes, and C. C. Olds. Summarizing Monte Carlo results in methodological research: The one- and two-factor fixed effects ANOVA cases. *Journal of Educational and Behavioral Statistics*, 17(4):315–339, Dec. 1992.

[6] M. Huang, A. Bavier, and L. Peterson. PlanetFlow: Maintaining accountability for network services. *ACM SIGOPS Operating Systems Review*, 40(1):89–94, Jan. 2006.

[7] F. Hui and P. Mohapatra. Experimental characterization of multi-hop communications in vehicular ad hoc network. In D. B. Johnson and R. Sengupta, editors, *VANET 2005, 2nd ACM international workshop on Vehicular ad hoc networks*, pages 85–86, New York, NY, USA, Sept. 2005. ACM.

[8] G. Iannaccone. CoMo: An open infrastructure for network monitoring — research agenda. Technical report, Intel Research, Cambridge, UK, Feb. 2005.

[9] G. Insolvibile. The Linux socket filter: Sniffing bytes over the network. *Linux Journal*, 86, Mar. 2001.

[10] G. Jourjon, T. Rakotoarivelo, and M. Ott. A portal to support rigorous experimental methodology in networking research. In H. Li, T. Korakis, P. Tran-Gia, and H.-S. Park, editors, *TridentCom 2011, 7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Heidelberg, Germany, Apr. 2011. ICST, Springer-Verlag Berlin.

[11] S. S. Kolahi, S. Narayan, D. Nguyen, and Y. Sunarto. Performance monitoring of various network traffic generators. In R. Cant, editor, *UKSim 2011, 13th International Conference on Computer Modelling and Simulation*, pages 501–506, Los Alamitos, CA, USA, Mar. 2011. IEEE Computer Society.

[12] Y. Koucheryavy, D. Moltchanov, and J. Harju. Performance evaluation of live video streaming service in 802.11b WLAN environment under different load conditions. In G. Ventre and R. Canonico, editors, *Interactive Multimedia on Next Generation Networks*, volume 2899 of *Lecture Notes in Computer Science*, pages 30–41. Springer-Verlag Berlin, Heidelberg, Germany, Nov. 2003.

[13] J.-Y. Le Boudec. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, Nov. 2010.

[14] S. Morley and M. Adams. Some simple statistical tests for exploring single-case time-series data. *British Journal of Clinical Psychology*, 28(1):1–18, Feb. 1989.

[15] S. Narayan, K. Brooking, and S. de Vere. Network performance analysis of VPN protocols: An empirical comparison on different operating systems. In X. M. Gu, J. P. Liu, V. E. Muhin, W. H. Peng, M. S. Qaiser, R. M. Shboul, Z. H. Wang, Y. W. Wu, Y. Zheng, and J. Zhou, editors, *NSWCTC 2009, International Conference on Networks Security, Wireless Communications and Trusted Computing*, pages 645–648, Los Alamitos, CA, USA, Apr. 2009. IEEE Computer Society.

[16] K. Park and V. S. Pai. CoMon: A mostly-scalable monitoring system for PlanetLab. *ACM SIGOPS Operating Systems Review*, 40:65–74, Jan. 2006.

[17] P. Primet, R. Harakaly, and F. Bonnassieux. Experiments of network throughput measurement and forecasting using the network weather. In H. E. Bal, editor, *CCGrid 2002, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 413, Piscataway, NJ, USA, May 2002. IEEE Computer Society.

[18] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar. OMF: A control and management framework for networking nestbeds. *SIGOPS Operating Systems Review*, 43(4):54–59, Jan. 2010.

[19] F. Schneider, J. Wallerich, and A. Feldmann. Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware. In S. Uhlig, K. Papagiannaki, and O. Bonaventure, editors, *PAM 2007, 8th Internatinal Conference on Passive and Active Network Measurement*,

volume 4427 of *Lecture Notes in Computer Science*, pages 207–217, Heidelberg, Germany, Apr. 2007. Springer-Verlag Berlin.

[20] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, RFC Editor, Fremont, CA, USA, Jan. 1996.

[21] A. Tirumala, L. Cottrell, and T. Dunigan. Measuring end-to-end bandwidth with Iperf using Web100. In *PAM 2003, Passive and Active Monitoring Workshop*, number SLAC-PUB-9733, Apr. 2003.

[22] M. Tsukada, J. Santa, O. Mehani, Y. Khaled, and T. Ernst. Design and experimental evaluation of a vehicular network based on NEMO and MANET. *EURASIP Journal on Advances in Signal Processing*, 2010:1–18, Sept. 2010.

[23] J. White, G. Jourjon, T. Rakotoarivelo, and M. Ott. Measurement architectures for network experiments with disconnected mobile nodes. In A. Gavras, N. Huu Thanh, and J. Chase, editors, *TridentCom 2010, 6th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Heidelberg, Germany, May 2010. ICST, Springer-Verlag Berlin.