Internet2/Google Summer of Code Final Report
Open Source μ-Measurements: Characterizing Machine Noise Impact on Network Delay Metrics

Cesar Marcondes (UCLA)
Mentor: Stanislav Shalunov (Internet2)

08/20/2005

Abstract

The project initial proposal was based on the Internet2 Project Idea: "Noise calibration for bulk transport tool". Basically, the project consisted on collect and analyze data to determine timing noise of packets sent across a network. It was defined in the context of the project that the noise to be measured would be obtained by comparing depart and arrive timestamps, minus the original signal, as a way to verify the impact of the noise. The noise, of course, would depend on the test environment, such that a variety of environments would work best. However, in all cases the network delay (the signal) needs to be known—otherwise, it becomes impossible to separate from noise before the characteristics of noise are known. So, a back-to-back environment was used. The most important task was to vary the operating system (different implementation port of the micro-measurement tools) and the load on the machines (it's the machines themselves that are the source of the noise; the rest, by definition, is signal). The purpose of this project was to provide input for building the Internet2 bulk transport tool. As a final note, it was developed and evaluated under the Google Summer of Code Project 2005.

Contents

1 Introduction **3**

2 μ-Measurements Tools **4**

3 Noise Characterization **7**

3.1 Back-to-Back Dispersion - Interarrival Packet Difference 7

3.2 Internal Queueing Delay 10

3.3 Inserting Machine Stress in Network Delay Measurements 12

4 References **15**

List of Figures

1 Hash-Based Measurements 5

2 Packet-Based Measurements 6

3 Inter Packet Departures 7

4 Inter Packet Arrival under Coalescing 7

5 Interarrival Packet Dispersion Under Coalescence 8

6 Interarrival Packet Dispersion Under Coalescence 9

7 Internal IP Queue Measurement 10

8 Internal IP Queue Under CBR and No Coalescing 11

9 Erlang PDF varying the bulk size (n. of stages) 12

10 No Stress Back to Back and Internal Queueing Measurement 12

11 Internal IP Queue Measurement under CPU Stress (a)(b)(c) 25%, (d)(e)(f) 50%, (g)(h)(i) 75% 13

12 Impact of Coalescing on Internal IP Queue Size 14

13 Internal IP Queue Measurement under CPU Stress (a)(b)(c) 25%, (d)(e)(f) 50%, (g)(h)(i) 75% 15

1 Introduction

The Internet has been upgraded successfully in the recent years, leading to a substancial boost of its own capillarity towards a next level ultra speed backbones with tenths of gigabit/s pipes in the core. On such environment, the common network bottleneck shifts from the wide-area to the end-host machine itself since the backbone can sustain plenty of bandwidth easily. Therefore, in order to keep up high processing performance, and sustain high network load as well, new network hardware mechanisms and operating systems designs have been proposed to tackle such problem using complicated tricks to offload the burden of processing millions of packets per seconds (as modern routers), reducing the likelihood of poor performance due to the very machine.

The main idea of this work is to gain insight about the influence of these new "offload" mechanisms on delay based measurements. In order to accomplish this, we focus mainly on micro measurement and characterization using instrumented open source kernel software based on Linux and FreeBSD. The major goal is to assess the influence of machine offload mechanisms and different machine processing/memory/IO load compositions over network delay measurements. In this document, we call any "change" of the supposed correct packet timestamping from the sender NIC card to the receiver application layer as "machine noise". Machine noise can be interpreted in different views but for our purposes, it represents errors/adaptation injected by the pair OS/HW, not network jitter. Therefore, for the whole set of experiments, we confined our tests only on a back to back machine basis.

In addition, this study can be seen as an important step towards achieve robust delay-based measurement algorithms in presence of offload mechanisms. For example, despite the improvement in capacity of the Internet, few transport protocols have shown the ability to obtain the huge residual capacity (gigabit/s) of these links. The most successful ones are TCP protocols like FAST, BIC, and TCP Westwood that accomplish high efficiency based on delay congestion control algorithms. It's easy to see the advantage of these advanced protocols compared to TCP NewReno since such delay based algorithms returns much more information about the path conditions than pure loss based algorithms. However, as we pointed out before, as the speed continues to raise, their own foundations on delay measurements can suffer a lot of difficulty on factoring out network delays (i.e. queueing delays). Perhaps, depending on the speed we are dealing with the offload mechanisms couldn't have a strong influence. Although, as the network reaches 1Gbps speeds, even a few μ secs error in an estimation could represent a fair amount of wasted traffic.

Finaly, following this new trends in high speed scenarios, in parallel to TCP advances, there is an idea to bulk transfer using rate-based algorithms over UDP that could reduce the burstiness of TCP algorithms and transfer huge demands of traffic even under small buffers, a very likely scenario in the future. However, as their TCP cousins, unfortunately they also suffer from the same problem, since they could be also based on the natural information-augumented delay measurements.

The rest of the report is organized as follows. In section II, we present our characterization tools discussing the underneath ideas for improved accuracy on the network measurements. In section III, we perform an extensive set of tests, fully characterizing the network/machine behavior at micro scale. In section IV, we present a simplified model of deconvolution of the the composed final packet dispersion measurement and an bulk arrival erlang model to obtain the original signal. Section V concludes the report.

2 μ-Measurements Tools

The tools described in this section were designed to measure the effect of machines impairments on network delay measurements. We developed this set of supplementary tools as pure open source software, as part of an initiative led by Google during Summer 2005. Our main goal was to capture fine granularity details of different traffic patterns as well as hardware mechanisms to alleviate high interrupt loads such as interrupt coalescing.

We chose as development platform a variety of operating system: mainly Linux (2.4-branch), Linux (2.6-branch) and FreeBSD 5.4. Such multitude of operating systems is a powerful way of generalize results and test under different computer architectures. It's also a good way to openly allow other developers to perform further estimation and filtering work in such open based kernel systems. The choice of two Linux kernels rely on the increase on Linux most advanced kernel banch to support more network offload mechanisms to scalable high-speed networking. One example of this trend, it's the the newest Linux 2.6 using NAPI that instead of per-packet interrupt scheduling and processing introduces the concept of one scheduled interrupt per pool. There are even more recent movements inside the Linux community to offload the kernel and rely heavily on the NIC cards to support 10bps of bandwidth.

One of the first assumptions of our micro-measurement tools was to assure that a quite accurate precision could be reached. Therefore, in all our open source modifications of the kernel, we heavy use the Intel-specific TSC register. Basically, TSC is a instruction counter that has the same stable features of the CPU oscillator. It's usually used for operating system profiling and high performance end tests. It has an astonishing resolution of 0.3271 nano seconds, since in our case, we were using 3.06GHz machine. And additionaly we can add that the register read is extremely fast and smaller than 50 nanosec. In terms of bottleneck computer architecture we used 3 Intel Xeon machines: 1 Dual (2 CPU) 3.06 Ghz Xeon with Cache 512KB L2/1GB L3, 1GB RAM and PCI-X Architecture and the other 2 Dual (only 1 CPU) 3.06 Ghz Xeon with Cache 512KB L2/1GB L3, 1GB RAM and PCI-X Architecture. The main advantage of using such powerful architectures is that we can stress the network conditions even further instead of cutting short of other resources such as I/O bus and memory-CPU bus.

We developed in Linux and FreeBSD, three different schemes increasing in software complexity. The first scheme, the most naive one relies simply on kernel debugging features been generated directly to a separate RAMdisk in order to obtain the results of back-to-back packet dispersions and internal queueing delays, we called this scheme log-based. The main advantage is that it was easy to get started, very simple and when generating to the RAMdisk in these powerful machines. However, even though we are using only main memory to store the timestamps, when the number of timestamps - proportional to the number of packets is high, we might suffer some page faults since the amount of used memory can be larger than the cache memory.

A more sophisticated method reducing the usage of RAM memory, that tries to fit the whole arrival time histogram in one single memory page in cache and therefore removing the extra IO operation was the so-called hash-based measurement mechanism. The reasoning behind it is to mimic an online real-time histogram as the packets reach the specific parts of software using a hash table to bucket sort according to the interarrival time. The following Figure 1 presents the mechanism based on the mask of a certain number of bits. The measurement resolution can be configured by the user using a sysctl variable, but the amount of memory to be used is always the same. Using the new measurement reset system call the user can clear the hash table and retrieve after the measurement, the correspondent histogram results.

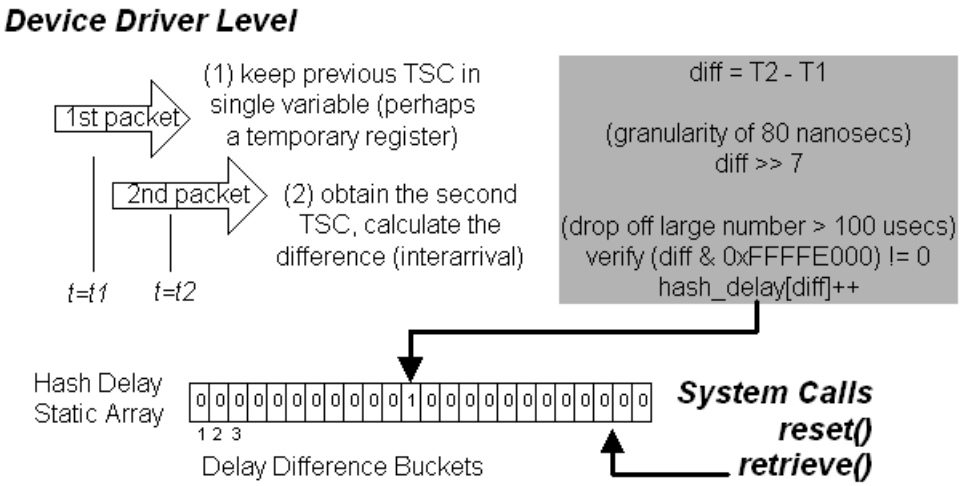


Figure 1: Hash-Based Measurements

The last method explore more correlated information among packets than the previous two methods, it's called packet-based measurement mechanism and it's a modification of the kernel to write inside the packet small timestamps as it passes through the main internal queues. In the case of the Linux kernel, there are essentially 3 main queues, the socket queue (nearest to the application itself), the ip queue usually the own that grows the most because of multiplexing of TCP/UDP/IP packets and the NIC/ring queue (smallest memory mapped IO area of the packets copied by DMA). We placed our code in the respective calls of every other part of the Unix kernel packet path to observe how they behave and we send a 1500 bytes UDP packet using port 5003 with the payload area zero so that the timestamps can be placed in the respective offsets, see more details in Figure 2.

One of the main motivation ideas of this scheme was to use a similar functionality already proposed in the ICMP-replacement protocol "IPMP" (Internet Measurement Protocol), basically the protocol allows probing packets to grow and trace the routers and the timestamp delay (queue size) as they passes through. The approach here is slightly different though, the idea is timestamp the micro queues inside the kernels instead of abstract the full node as a single queue. The advantages of this scheme are basically study of packet correlation, for example how packet pairs are being queued, how much, what is the empirical service rate, if it exists any packet dispersion compression due to the internal queues or expansion, finally which internal queue of the OS is the most affected by a certain CPU/Mem/IO load mix or even offload mechanism. We manage to assess the impact of every packet rewriting and the value was a constant of 0.13 micro seconds per rewrite, a total of 0.78 micro second of overhead per packet. In the experiments section, this extra overhead was responsible for a cap (packet loss rate) of 2.4% in 400Mbps, 15% in 500Mbps, 30% in 600Mbps, 53% in 700Mbps and 74% in 800Mbps, even though some packets went through.

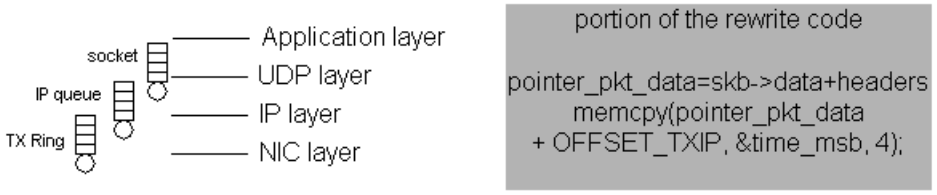


Figure 2: Packet-Based Measurements

3 Noise Characterization

The evaluation of "machine noise" was done in separate phases (to accurately separate and dissecate each one of the noise sources). One of the initial phases consisted of back-to-back packet dispersion (interarrival difference packet times) measurements in order to understand the behavior of the original source traffic pattern in the presence of interrupt coalescing. In these measurements, we obtained the interarrival time in the device driver software at the receiver machine connected directly to the sender machine. The sender machine on the other hand, generate a variety of offered network load traffic and traffic patterns using a modified thrulay version. All tests were performed 4 times to guarantee statistically significance, also the duration of each experiment was about 30 seconds. As a future step, we intend to leverage this dataset for future collaborative studies.

At this step, in order to understand the interrupt coalescing hardware mechanism, we need to explain in some detail the configuration and how this mechanism works in the Intel 1000MT NIC Server Adapter. The example to be used is the FreeBSD e1000 latest driver whose default configuration is RX Absolute Interrupt Delay = 66 μsecs and TX Absolute Interrupt Delay = 66 μsecs as well, that stand for an absolute interrupt cycle of total 132 μsecs, any packets that arrive inside this interval are bunched together. The other mechanisms presented in the Intel documentation, such as interrupt delay between last packet arrival and the interrupt, were not used in our experiments.

3.1 Back-to-Back Dispersion - Interarrival Packet Difference

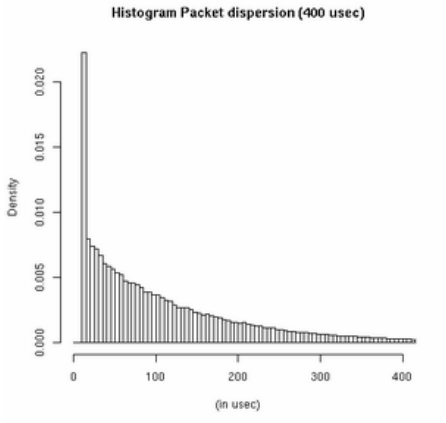


Figure 3: Inter Packet Departures

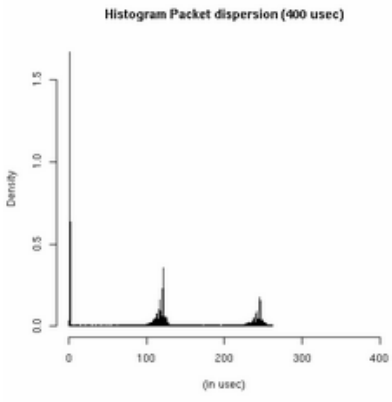
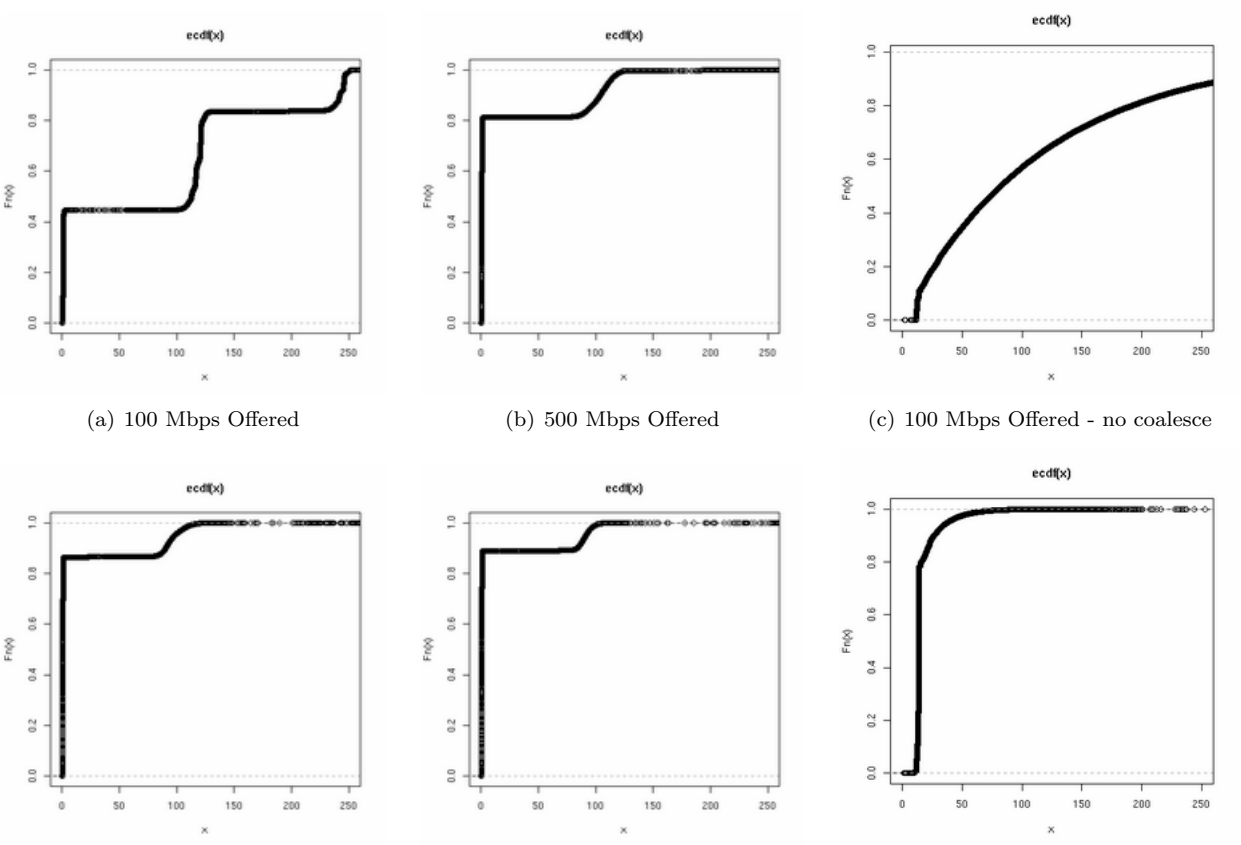


Figure 4: Inter Packet Arrival under Coalescing

The first set of experiments was done using FreeBSD 5.4 along with the instrumentation code. As it

can be seen the original traffic pattern [Figure 3] was a poisson source (exponential interdeparture decay). However, in the receiver side we can see that the signal was totally disturbed by the interrupt coalescing and three strong modes happen whenever the traffic offered load is around 100Mbps [Figure 4].

As the offered load increases, the bulk size of an absolute interrupt increases, as well the probability of a certain packet pair be bunched together [Figures 5 (a), (b), (d), (e)]. Therefore, we can see from the Empirical CDF (Cumulative Distributions) such behavior of the interrupt moderation and without coalescing [Figures 5 (c) (f)]. In the following Figure 5 the x-axis (variable "x") represents the interarrival delay in μsecs, and observe that as the load increases the modes do not change, they still are around 1-2 in μsecs and a second one around 132 μsecs.



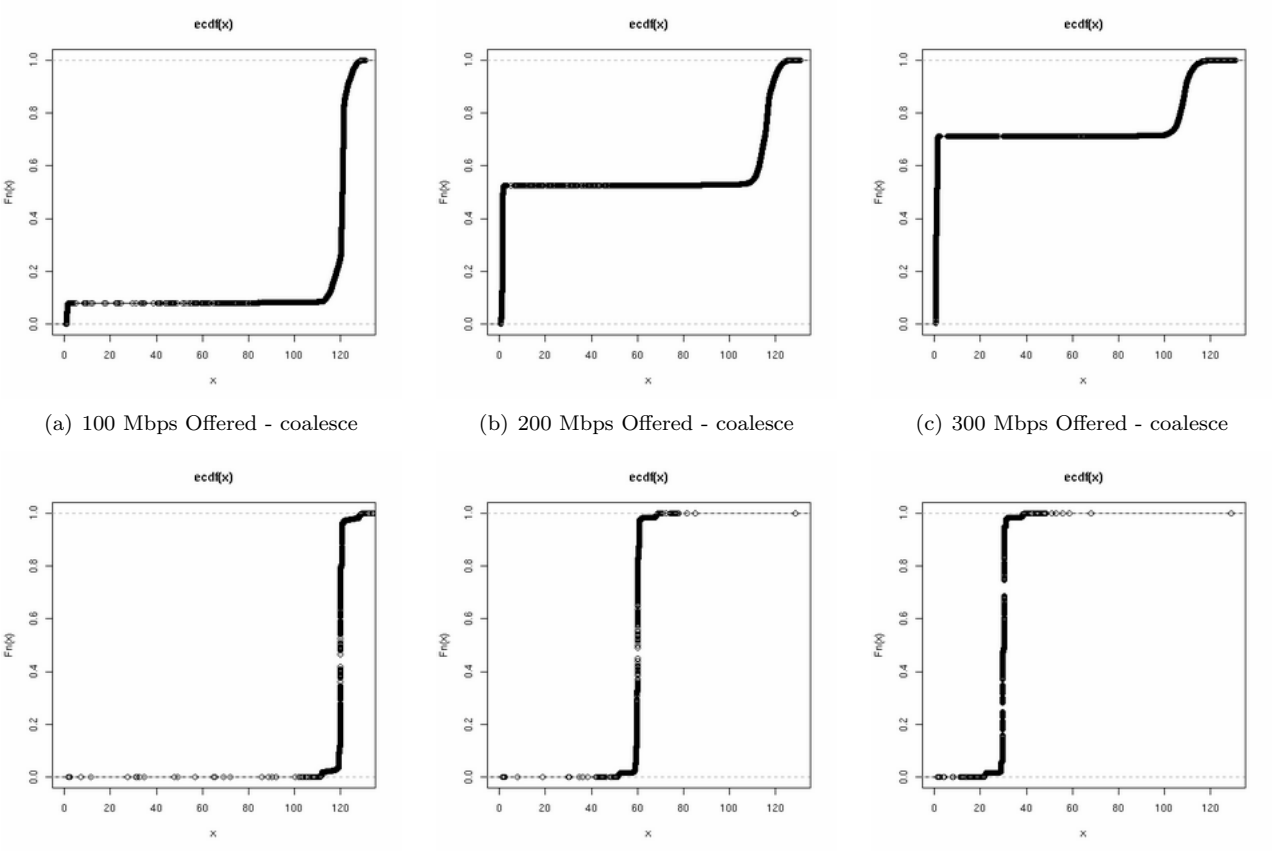
(a) 100 Mbps Offered (b) 500 Mbps Offered (c) 100 Mbps Offered - no coalesce

(d) 700 Mbps Offered (e) 1000 Mbps Offered (f) 700 Mbps Offered - no coalesce

Figure 5: Interarrival Packet Dispersion Under Coalescence

After this initial poisson interarrival characterization, we tried also to further measure the interarrival

packet dispersion whenever we have a pure CBR source, this way we could deterministicly measure the bulk size with interrupt coalescing and measuring the precision of the tool without coalescing.



(a) 100 Mbps Offered - coalesce (b) 200 Mbps Offered - coalesce (c) 300 Mbps Offered - coalesce

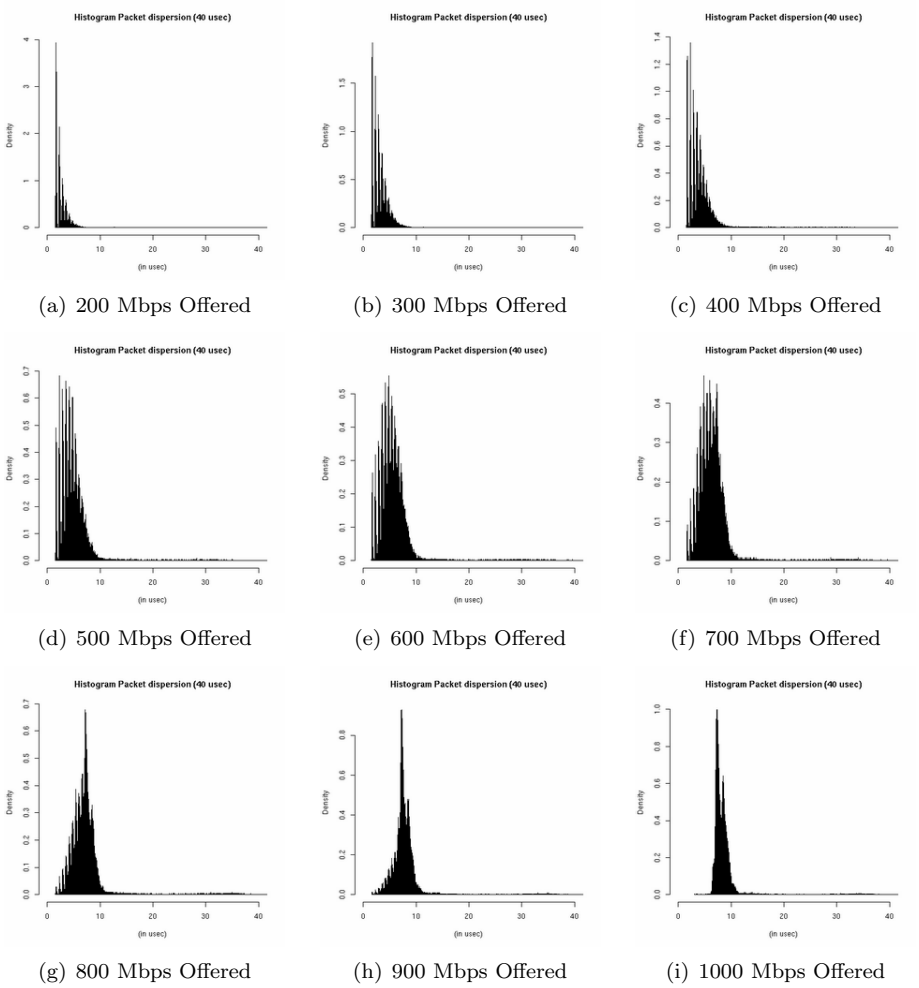
(d) 100 Mbps Offered - no coalesce (e) 200 Mbps Offered - no coalesce (f) 300 Mbps Offered - no coalesce

Figure 6: Interarrival Packet Dispersion Under Coalescence

If one compare carefully the poisson and CBR pattern under coalescing regime will see that the first one has three modes and the second only two modes, this happen because while the poisson source generates interdepartures range widely from few μsecs to more than 400 μsecs (figure 3), it triggers two absolute interrupts every cycle of a second. While the CBR fixed on the 120 μsecs has the majority of packet coalesced together in the second absolute triggered interrupt (figure 6).

3.2 Internal Queueing Delay

The second experiment phase focused on the measurement of the internal ip queue and we observe additionally the impact of two factors (source traffic pattern and coalescing) on the size and variance of the ip queue alone. Let's start by looking at Linux 2.4.20 IP queue behavior.



(a) 200 Mbps Offered (b) 300 Mbps Offered (c) 400 Mbps Offered

(d) 500 Mbps Offered (e) 600 Mbps Offered (f) 700 Mbps Offered

(g) 800 Mbps Offered (h) 900 Mbps Offered (i) 1000 Mbps Offered

Figure 7: Internal IP Queue Measurement

The first set of results are 30 seconds period samples under poisson traffic generated from one machine to the other (back to back) measuring only in the receiver the time that a packet reaches the NIC driver (netif_rx in linux 2.4.20) until the time it exits the ip queue and it's copied into the UDP socket (udp_rcv

in linux 2.4.20). We used only single user mode in both machines to remove any further noise on this scenario.

It is important to notice that the ip queue is extremely small (in terms of delay), only around 1 μsecond and at our micro level measurement, the two "read operations and the 64 bit difference of the timestamping" from the TSC register has a special role to create some evenly distributed spikes in the histogram measurements. The figure 7 above shows all the ip queue histograms (per packet overhead delay due to ipqueue) varying the offered load from 200Mbps to 1Gbps.

One can say that at 200Mbps offered load, the bulk size from the absolut interrupt coalescing is not that big to cause the queue to grow as much as 3 μsecs in average, and all packets suffer equally this queueing delay - not much variance, so this scenario imply that there is not much impact in the application layer delay-based measurement.

Another issue here, using the poisson process source in this scenario, is that the bulk size is going to be non-deterministically distributed since the poisson cycle is measured in seconds and the interrupts happen at 130 μsecs, such uncertainity on the number of packets processed per one single burst, generates more variability in this ip queue than the CBR case as in Figure 8. As the load grows up, the absolute NIC interrupt interval is filled with more packets and a full bulk of packets arrive simultaneuosly causing the ip queue to grow as well.

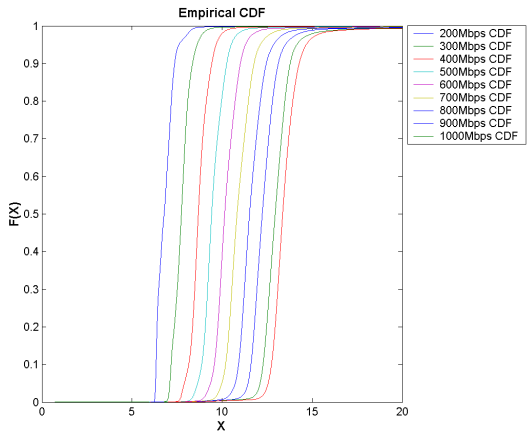


Figure 8: Internal IP Queue Under CBR and No Coalescing

Continuing our discussion about Figure 7 as the load increases the overhead per packet increases proportionally to the load as well as the queue variance (order of 1-10 μsecs), average 5 μsecs. Such variability

could affect delay-based measurements but the measurement pattern was already destroyed by the interrupt coalescing.

However, this internal queue give us an insight of the bulk size distribution since the embedded probability process that we are observing here seems quite similar to a bulk arrival system or M/Er/1 markovian process where the stages represent the sustained bulk size from the interrupt coalescing part (Figure 9). We can observe that As the offered load reaches the maximum of 1Gbps, it's clear that the queue is full and there is no left space since the variance reduces dramatically around 10 μsecs (in figure 7).

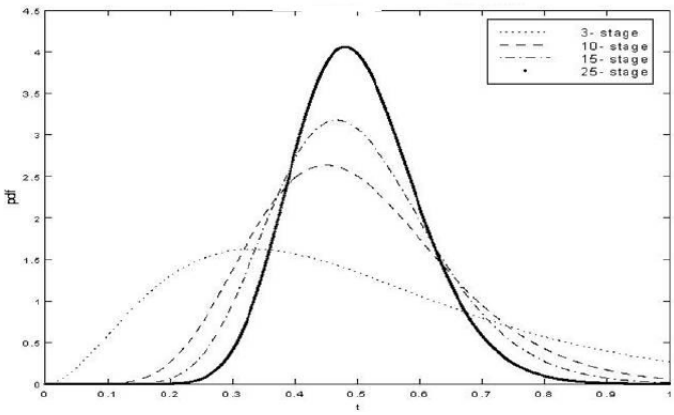
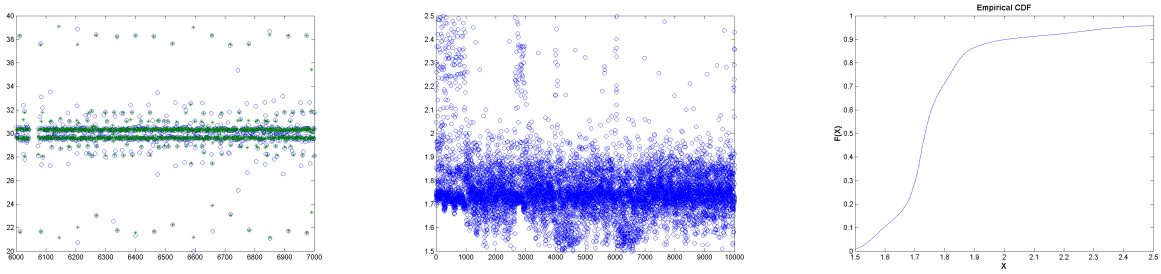


Figure 9: Erlang PDF varying the bulk size (n. of stages)

3.3 Inserting Machine Stress in Network Delay Measurements

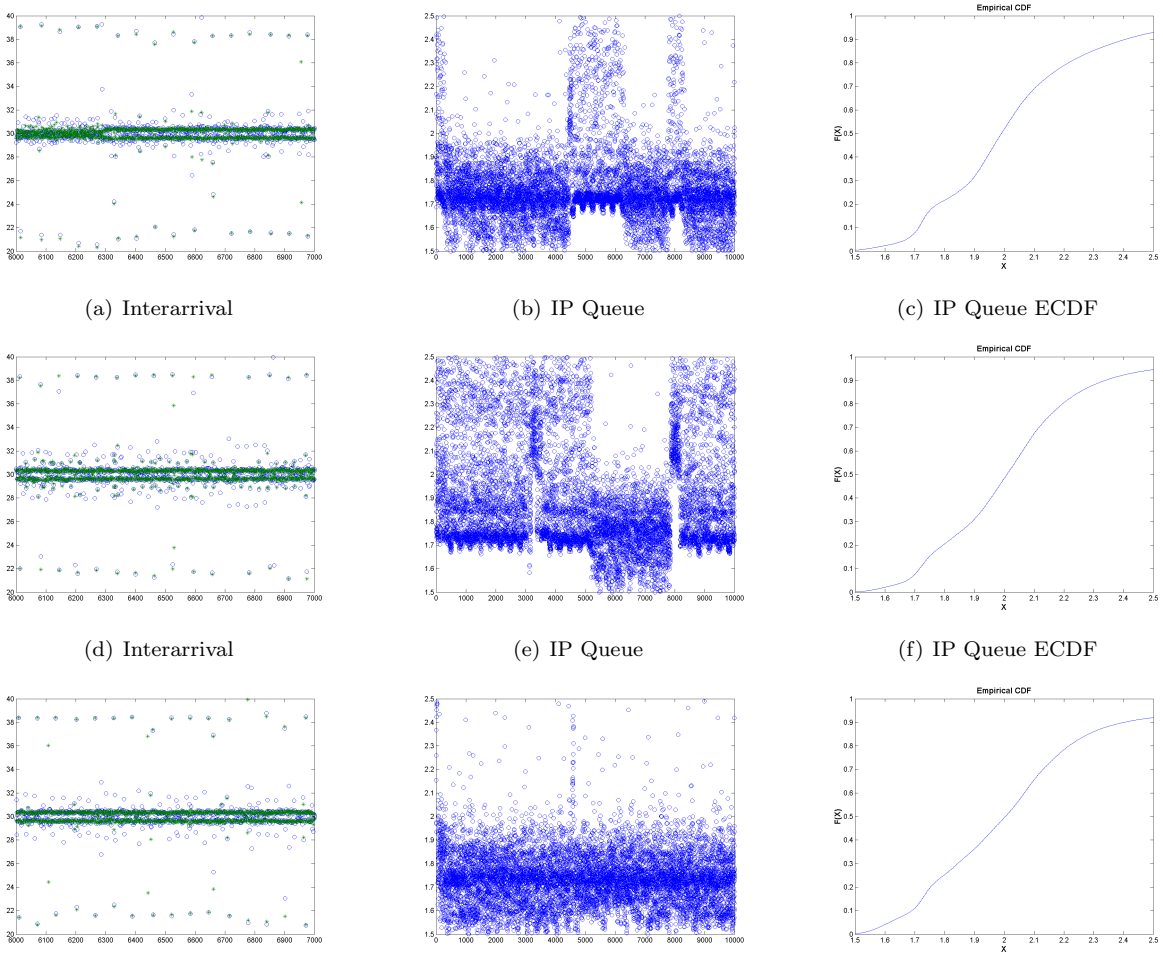


(a) NIC vs UDP timestamp (b) IP Queueing Receiver (c) IP Queueing ECDF

Figure 10: No Stress Back to Back and Internal Queueing Measurement

In this section, we are going to present results whenever we apply some stress in the system (context

switches due to scheduling, strong I/O bus contention, etc). We used the "packet-based measurement" scheme to measure, each internal queue in a single pass (socket buffer, ip queue buffer and transmission ring). Additionally, in order to this work be unbiased from the interrupt coalescing point of view, all the following experiments were under CBR traffic and no coalescing regime. So, only machine noise and the original signal. Since, there are a quite amount of data, we are going to discuss on top of the 400Mbps example only.



(a) Interarrival (b) IP Queue (c) IP Queue ECDF

(d) Interarrival (e) IP Queue (f) IP Queue ECDF

(g) Interarrival (h) IP Queue (i) IP Queue ECDF

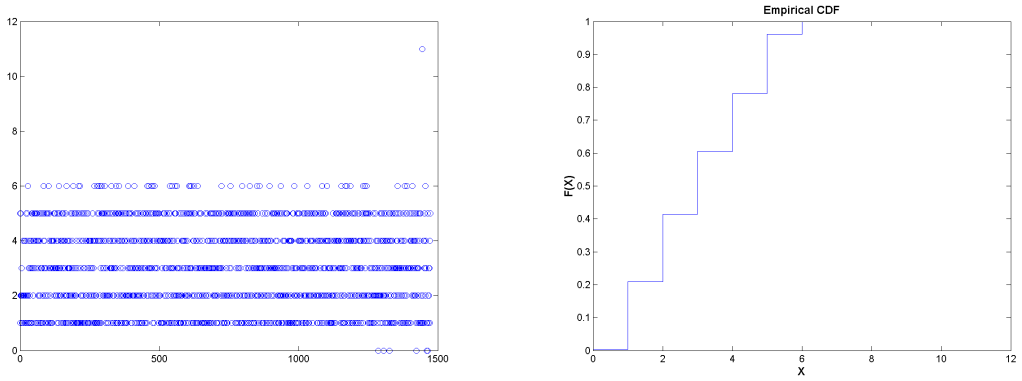
Figure 11: Internal IP Queue Measurement under CPU Stress (a)(b)(c) 25%, (d)(e)(f) 50%, (g)(h)(i) 75%

We start by presenting the baseline results of NO STRESS. In Figure 10 we can see the Linux Internal queue is quite stable near 1.7 μsecs. Most of this "no stressed" queue variation is about 0.2-0.3 μsecs, in other words, the timestamping TSC variation error level.

As we add the stress by inserting one, two and three CPU Intensive processes, respectivelly responsible for 25%, 50% and 85% of CPU utilization we observe the network delay measurement remains the same but in microscopic scale, the internal queue varies a lot more (Figure 13). It's a good to remember that Linux has the network interrupt as the most priority interrupt in the whole OS. For this stressed measurement we used the tool STRESS from the website http://weather.ou.edu/ apw/projects/stress/.

As the CPU stress increases, the linux internal queue variation has a trend of increase but it's the amount of increase can barely be seen from the Empirical CDF graph. And therefore we can observe that under Linux, no CPU intensive process overcome or disrupt a network service and its precise back to back interarrival measurements.

Just to point out that whenever we are dealing with non-coalescing regime, we obtain less internal queue variation, in all analyzed scenarios the IP queue didn't grow, it was kept always in 1 packet. However, if we use CBR traffic and coalescing under 400Mbps we obtain a strange Empirical Queue Size Distribution (uniform!!! - in Figure 12).



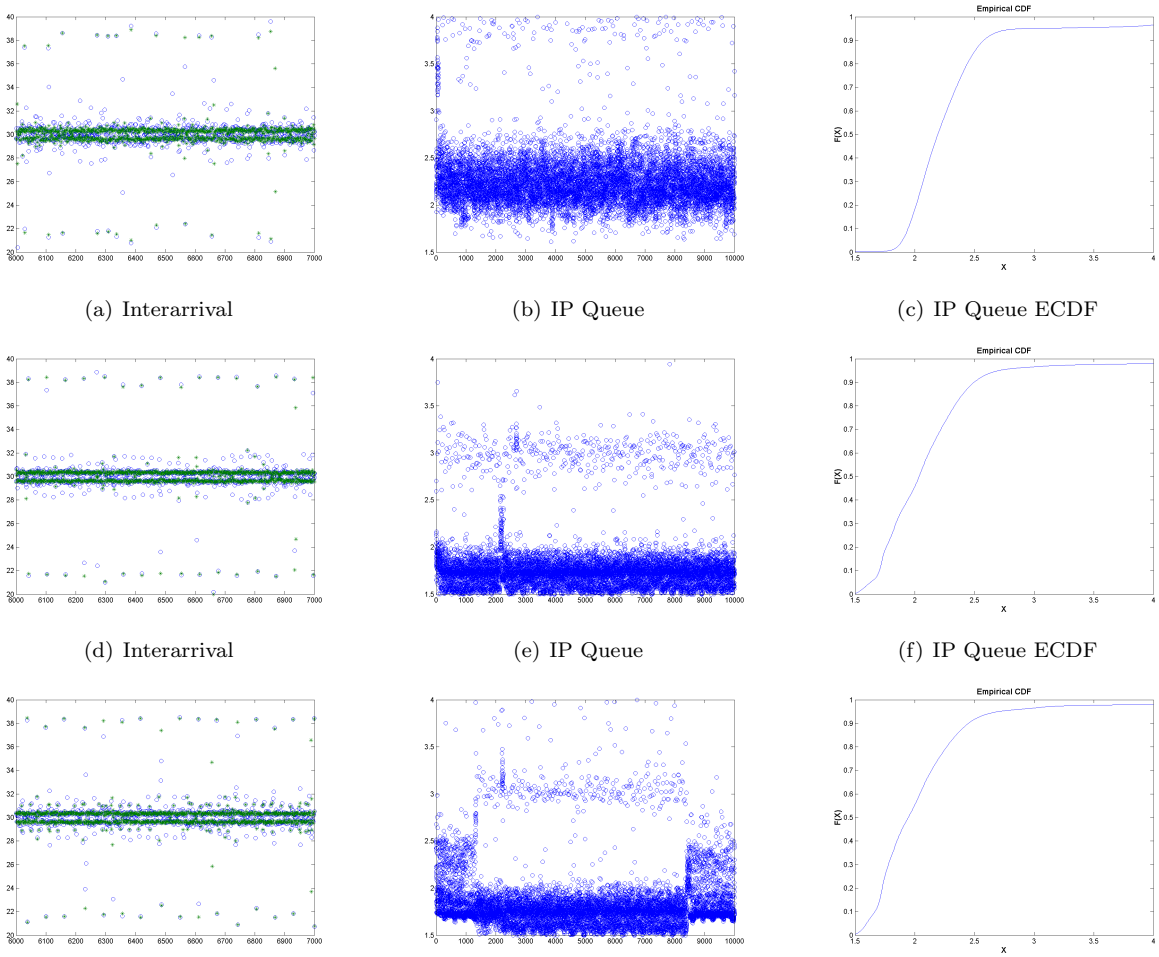
(a) CBR + coalescing at 500Mbps - Queue Size Time Series

(b) CBR + coalescing at 500Mbps - Queue Distribution

Figure 12: Impact of Coalescing on Internal IP Queue Size

The next figure shows the effect in the Linux Internal Queue of I/O intensive processes, again we used

the open source to generate 25%, 50% and 75% CPU utilization, but since we are dealing with I/O processes the kernel is the one that take care of obtaining data.



(a) Interarrival (b) IP Queue (c) IP Queue ECDF

(d) Interarrival (e) IP Queue (f) IP Queue ECDF

(g) Interarrival (h) IP Queue (i) IP Queue ECDF

Figure 13: Internal IP Queue Measurement under CPU Stress (a)(b)(c) 25%, (d)(e)(f) 50%, (g)(h)(i) 75%

4 References

Passive Calibration of an Active Measurement System

Design Principles for Accurate Passive Measurement

Evaluation and Characterization of Available Bandwidth Probing Techniques

System Capability Effects on Algorithms for Network Bandwidth Measurement

Towards Tunable Measurement Techniques for Available Bandwidth

Eliminating receive livelock in an interrupt-driven kernel

Effects of Interrupt Coalescence on Network Measurements

Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput

Analysis and simulation of interrupt overhead impact on OS throughput in high-speed networks

PC Based Precision Timing Without GPS