Experiments with Practical On-Demand Multi-Core Packet Capture

Marat Zhanikeev*
* Department of Artificial Intelligence,
Computer Science and Systems Engineering,
Kyushu Institute of Technology
Kawazu 680-4, Iizuka, JAPAN 820-8502
Email: maratishe@gmail.com

***Abstract*—This paper proposes and evaluates performance of an on-demand packet capture process in multi-core architectures. The multi-core on-demand process presented in this paper can handle higher packet throughput and is sufficiently flexible to support even complex per-packet processing logic.**

***Index Terms*—multi-core architecture, on-demand packet capture, online traffic analysis, packet processing overhead**

I. Introduction

High-performance packet capture on commodity hardware has recently become feasible, as long as the traditional *libpcap* technology is replaced with more efficient middleware like PF_RING [1]. Multi-core capture is a recent topic and does not have much existing literature [2]. The few existing studies do not offer a practical use for the technology. This paper studies *the utility* of the multi-core packet capture and specifically the possibility of adding and removing capture processes (cores) on demand. The latter technique has many practical uses like parallel traffic processing, on-demand anomaly detection, precision flow accounting, etc. Theoretically, the on-demand multicore environment makes it possible to mix shallow and deep analysis of traffic.

II. Related Works

At packet level, per-packet processing cost is often the issue, where efficient hashing can help alleviate the problem. Earlier study by this author showed that cost varies wildly depending on processing logic [3]. Packets are not always aggregated into flows. This author proposed a method which should capture QoS context at packet level without flow aggregation [5]. Also, analysis can be done at the level of IP address itself [6], also bypassing flows.

One specific sampling discipline – the context-based sampling – requires special attention. This author contributed to this topic by proposing several algorithms based on *batch sampling* [7][8]. Context awareness increases the per-packet cost but a multicore process can distribute the increased cost over multiple cores.

Multicore capture can solve two major problem. It can help avoid sparse sampling [9]. Also, all methods which use packets directly – like the anomaly detection method in [4] – can also benefit. Basically, any heavy-duty packet processing should benefit from multicore capture [5][7][8].

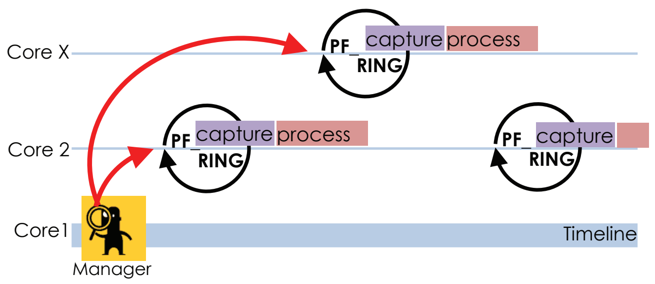

Figure 1: The on-demand packet capture process which exploits multiple cores on commodity hardware.

The idea of a multicore capture was first expressed is [1] by the creator of *pfring* himself. There are, however, surprisingly few studies on putting this functionality to practical use. The only study which is close in spirit to this study is in [2]. However, the findings in this paper are opposite to those in [2] in that this paper finds that there is very little overhead from introducing additional cores.

III. The On-Demand Capture Process

Fig.1 depicts the basic idea behind the on-demand capture process. For each *capture request*, Manager spawns a separate capture process on a free core. At spawning time (forking, actually), the on-demand thread is configured with its *target* – a selection rule for incoming packets. Each on-demand capture is split into *capture* and *process* phases. □

Note that this design is open to several optimization problems. For example, scheduling of on-demand threads in such a way as to optimize utilization of cores can be one such optimization problem.

IV. Experimental Setup and Target

An Intel i7 3.2GHz box with 4 cores and 8 threads (CPU multithreading) is used for the tests. Built-in NIC is used as is. Kernel module of *pfring* is precompiled and loaded into the kernel prior to running experiments.

Fig.2 shows how the testbed is designed. 1Gbps environment is supported by using 1GbE hub and making sure that all machines have 1GbE NICs. Three separate machines are used as traffic generators. While it is possible to replay packets from

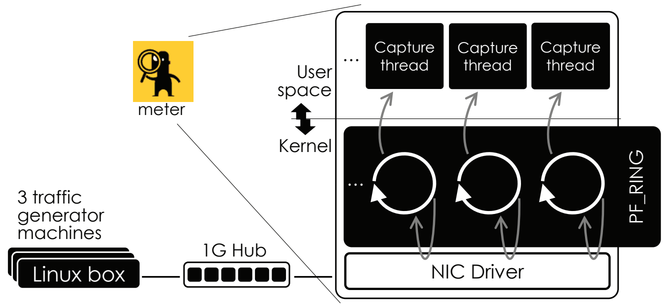

Figure 2: The experimental setup used to get results for this paper. The nominal capacity of the network is 1Gbps.

an existing trace [10], this author opted for well-controlled uniform workloads. In each run all traffic generators send as many packets as possible (no sleep in sending loop).

One experimental run is conducted as follows. All processes start on command from the *Meter*, only *pfring* manager is started by the main script locally. For each run, the main script collects data from all the machines and wraps it into a single file. Over 2500 runs each 10s long were conducted.

Two experiments were conducted. The first measures per-core capture throughput and the load (CPU utilization) it inflicts. The target is to see the relative effect from running multiple cores verses the traditional case of a single core. The second measures *reaction time* of on-demand capture – referred to as *thread lag* – between the intended and the actual starting time on a new on-demand capture process.

V. Experimental Results

Fig.3 is a 3D visualization of the results (2d + bullet size). Each data point in the plots is one experimental run.

The rightmost plot visualizes capture throughput across various packet sizes. There is a well-pronounced exponential trend where smaller packet size greatly increases the capture throughput. The trend saturates at 500-byte packets and about 150kpps (150k * 500 * 7 $\geq$ 500Mbps). There are rare cases when 150kbps is registered for larger packets (closer to 1Gbps) but most cases are below 100kpps. Roughly the same throughput is achieved with small packet size (700kpps, 64 bytes). The effect of multicore capture on throughput is found in the left part where there is a clear trend showing that **fewer cores can achieve hig her throughput when packet size is small**.

The other two plots show how load, measured as the sum of CPU utilizations across all cores, is affected by the multicore capture. The plots in fact show **the lack of any effect** as bullets are mixed roughly uniformly across the plots. The leftmost plot repeats the above trend in which higher packet throughput is achieved by fewer cores. In the vertical dimension, there is no trend, which means that **multicore capture does impose a major penalty on CPU load**.

Table.I shows the setup for this second experiment. Note that interval between threads is referred to as *thread gap*. The

TABLE I: Parameters used for experiments on rapid on-demand capture.

Parameter name	Parameter values	Comments
interval	100,200,500,1000,2000,5000, 10000,50000,100000,200000, 500000 (microseconds)	Interval the manager thread uses to spawn capture threads.
packet size	between 100 and 1200 bytes with step 100	Used for background traffic
threads	between 2 and 7 with step 1	Number of capture threads, spawned by the manager thread.

objective is to measure *thread lag* for various combinations of *packet size*, *number of cores*, and *thread gap*.

Fig.4 shows the results in two dependency combinations – thread lag against packet size and thread lag against the configured thread gap. Note that each curve represents data collected for one of the spawned threads, where "Thread 3" means *the third spawned thread*. Both plots in Fig.4 show high volatility as data points – averages of multiple experiments under a given configuration – in some rare cases depart from the overall trend. The high level of volatility is also due to the presentation method itself where each curve represents only one pair of threads (as in gap/lag between 5th and 4th threads). The overall trend can still be confirmed visually. The upper plot in Fig.4 shows that **thread lag does not depend on packet size**. The average lag is around 10-15ms (positive), regardless of which thread pair is selected.

The lower plot of Fig.4 also shows roughly the same performance for all curves (jitter aside), which, as was previously discovered, hints at the lack of dependency on the number of threads in a multicore process. However, **there is dependency of thread lag on thread gap** itself (beware the log scale). The curves start to increase starting from 1ms gaps (10^4) and keeps increasing in a piecewise linear trend. Between 10^4 and 10^5 the lag is around 30ms but is doubled for 500ms gaps.

The simple reading is that for half a second interval between threads, one can expect about **15-20% lag** from the intended/scheduled starting time of packet capture. This result also confirms that *pfring* architecture incurs internal delays (in kernel space) when creating new *pfrings* next to heavily utilized existing ones.

VI. Conclusion

This paper is the first to propose the concept of on-demand packet capture in multicore architectures. Evaluating the performance of a simple implementation of this concept was the main objective of this paper. The multicore on-demand capture has become readily available in recent years as commodity hardware with 4-8 cores is now commonplace.

It was discovered that the multicore process has a *choking point*. In the presented testbed the choking point was found only at the smallest packet size – 64 bytes – where it was discovered that fewer cores could capture more packets on average. No correlation was found between CPU load and the number of

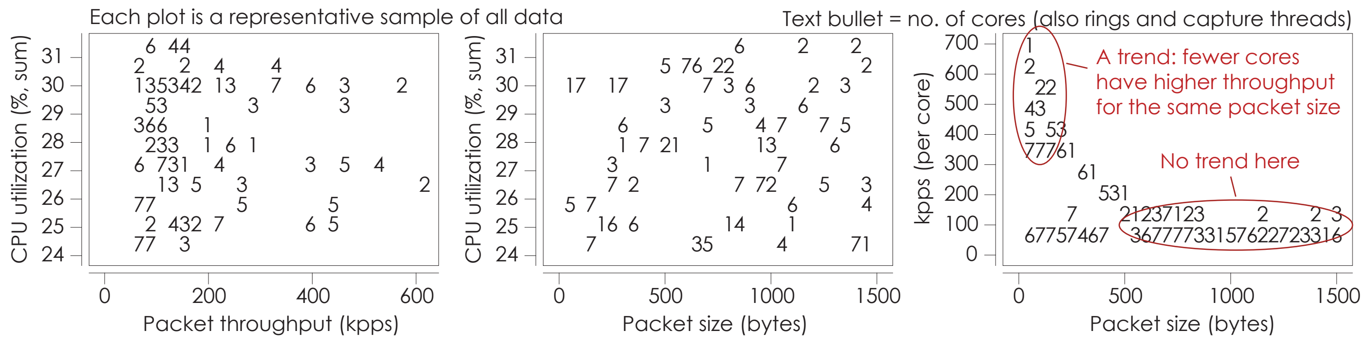

Figure 3: Results for Experiment A. Overhead from running multiple capture threads in parallel is non-existent. Note that in order to increase information density, bullets directly represent the number of cores, i.e. the number of capture threads running in parallel.

cores, regardless of packet size. This speaks in favor of the efficiency of the ring architecture advertised by PF_RING [1].

It was found that on average about 10-15% lag is to be expected for a newly spawned core, which means that the new thread starts capturing packets after a small lag. This lag has to be taken into consideration if the on-demand process is spawned to capture a specific flow or a group of flows at a given time. Once new rings are created, they do not differ in performance from preceding rings, as was confirmed by the results in this paper.

In future work, the on-demand capture process will be used as part of several methods developed by this author for realtime analysis of traffic, where the multicore architecture can offer considerable boosts both in efficiency and functionality.

References

[1] Luca Deri, "Modern Packet Capture and Analysis: Multi-Core, Multi-Gigabit, and Beyond", Internet Measurement (IM) Tutorial, 2009.

[2] M.Schultz and P.Crawley, "Performance Analysis of Packet Capture Methods in a 10 Gbps Virtualized Environment", 21st International Conference on Computer Communications and Networks (ICCCN), Munich, Germany, pp.1–8, August 2012.

[3] W.Xie, M.Zhanikeev, and Y.Tanaka, "Processing Overhead in IP Traffic Analysis", IEICE Communications Society Conference, No.BS-7-36, pp.S115–S116, September 2010.

[4] M.Zhanikeev, Y.Tanaka, "Lightweight Traffic Monitoring and Analysis Using Video Compression Techniques", Management Enabling the Future Internet for Changing Business and New Computing Services, Springer LNCS vol.5787, pp.92–101, September 2009.

[5] M.Zhanikeev, R.Yamamoto and Y.Tanaka, "Capturing QoS Context by Alternative Flow Monitoring in Clouds", IEICE General Conference, No.BDS-1-2, pp.S128–S129, March 2012.

[6] M.Zhanikeev, Y.Tanaka, "A Framework for Detection of Traffic Anomalies Based on IP Aggregation", IEICE Transactions on Information and Systems, vol.E92-D, no.1, pp.16–23, January 2009.

[7] M.Zhanikeev and Y.Tanaka, "Control over Precision of Flow Volume Sampling using Random Batch Sampling", IEICE Technical Report on Network Systems (NS), vol.112, no.463, pp.107–112, March 2013.

[8] M.Zhanikeev, Y.Tanaka and R.Yamamoto, "Alternative Packet Sampling for Improved Fairness and Function", IEICE General Conference, No.BS-3-4, pp.S7–S8, March 2012.

[9] Sardar Ali, Irfan Ul Haq, Sajjad Rizvi, Naurin Rasheed, Unum Sarfraz, Ali Khayam, and Fauzan Mirza, "On Mitigating Sampling-Induced Accuracy Loss in Traffic Anomaly Detection Systems", ACM SIGCOMM, 2010.

[10] D.Van, M.Zhanikeev and Y.Tanaka, "Effective High Speed Traffic Replay Based on IP Space", 11th International Conference on Advanced Communication Technology (ICACT), Phoenix Park, Korea, pp.151–156, February 2009.

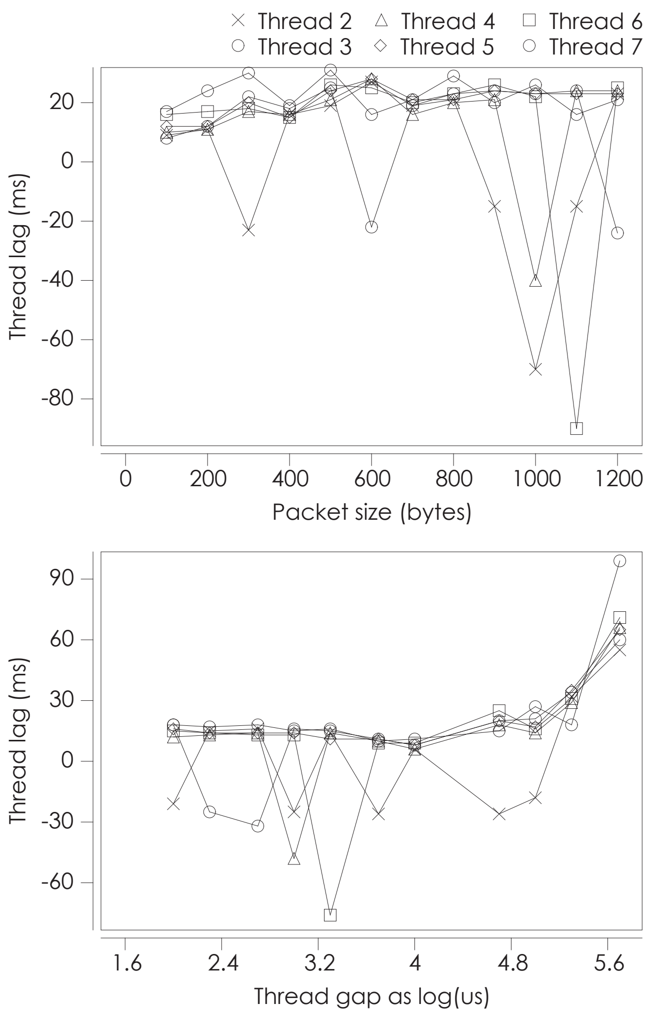

Figure 4: Results on the lag when spawning additional capture threads on demand.