In Procs. of the 6th International Symposium on High Performance Computing (ISHPC-VI),

Higashikasugano, Nara City, Japan, September 2005

Workload Characterization of Stateful Networking Applications

Javier Verdú[1], Mario Nemirovsky[2], Jorge García[1], and Mateo Valero[1,3]

[1] Departament d'Arquitectura de Computadors, UPC
Barcelona, Spain
[2] Consentry Networks Inc.
Milpitas, CA, USA
[3] Barcelona Supercomputing Center
Barcelona, Spain
{jverdu,jorge,mateo}@ac.upc.edu
mario@consentry.com

Abstract. The explosive and robust growth of the Internet owes a lot to the "end-to-end principle", which pushes stateful operations to the end-points. The Internet grow both in traffic volume, and in the richness of the applications it supports. A whole new class of applications requires stateful processing.
This paper presents the first workload characterization of stateful networking applications. The analysis emphasizes the study of data cache behaviour. Nevertheless, we also discuss other issues, such as branch prediction, instruction distribution and ILP. Another important contribution is the study of the state categories of the networking applications. The results show an important memory bottleneck that involves new challenges to overcome.

Keywords - Flow, state, non-additional-data, stateless, stateful.

1 Introduction

The explosive and robust growth of the Internet pushes stateful operations to the end-points. The Internet grow both in traffic volume, and in the richness of the applications it supports. The growth also brought along new security issues. The sophistication of attack methods is progressing rapidly.

The attackers take advantage of stateless firewalls that cannot look beyond the single individual packet while inspecting network traffic. The attacks may be spread out in several packets, such as inter-packet signature, or even may be undetectable with signature-based detection systems, such as portscan, unknown

attacks, or zero day attacks [7, 8]. Additionally, stateless Network Intrusion Detection Systems (NIDSs) may be overwhelmed with the Snot [5] and Stick [2] attacks. These attacks work by generating packets that are expected to trigger alerts by the NIDS. Therefore, more complex firewalls, that keep track of the processed packets, are being developed in order to catch these new attacks.

Another area of stateful applications is network monitoring, such as the commercial tool, called Cisco NetFlow [3], and the publicly available application, called Argus [4]. They are able to record important information along different granularity levels. From flow level, such as number of packets of a given flow, up to user level, such as where do the users go on the network.

The development of network processors is focused on overcoming the time constraints of both network line rates and networking applications workloads, mainly on the fast-path. Internet traffic continues to grow vigorously close to 100% per year [17] and consequently the traffic aggregation level reflects an incremental trend. On the other hand, concerning the statefulness of the application, there are more flow states maintained and more complex applications generate larger states. Consequently, the memory capacity requirements become a bottleneck.

This paper presents the first workload characterization of stateful networking applications. The analysis emphasizes the study of data cache behaviour. Nevertheless, we also discuss other issues, such as branch prediction, instruction distribution and ILP. Another important contribution is the study of the state categories of the networking applications. Our conclusions show that stateful applications present an important memory bottleneck that involves new challenges to overcome. We demonstrate that the basis of our conclusions can be generalized to other new stateful programs.

The rest of this paper is organized as follows. Section 2 provides related work on current networking benchmark suites and their workload characterizations. In Section 3, we present the benchmark selection, the traffic traces, and the methodology to perform the evaluation. The workload characterization is presented in Section 4. Finally, we conclude in Section 5.

2 Related Work

Benchmarking NPs is complicated by a variety of factors [9], such as emerging applications that do not yet have standard definitions. There is a high interest and an ongoing effort in the NP community to define standard benchmarks [15].

Several benchmarks suites have been published in the NP area: CommBench [25], NetBench [14] and NpBench [12]. Wolf et al. [25] present the CommBench benchmark suite: a set of eight benchmarks classified in Header Processing Applications (HPA) and Payload Processing Applications (PPA). The suite is focused on program kernels typical of traditional routers. The workloads are characterized and compared versus SPEC benchmarks.

Memik et al. [14] present a set of nine benchmarks, called NetBench. The authors categorize the benchmarks into three groups, according to the level of

networking application: micro-level, IP-level and application-level. The workloads are compared versus MediaBench programs.

Lastly, Lee et al. [12] propose a new set of ten benchmarks, called NpBench. It is focused on both control and data plane processing. In this case, the benchmarks are categorized according to the functionality: traffic management and quality of service group (TQG), security and media processing group (SMG), and packet processing group (PPG). The study of the workloads is compared with the CommBench workloads.

All the above benchmarks are no stateful applications, since they do not keep track of the previous processed packets. An IDS called Snort [1], which is included in the NetBench suite, although it is not included in the original paper [14], is the single application that presents stateful features. Moreover, depending on Snort's configuration, the statefulness of the processing may vary a lot. There are several publications that present studies about Snort [19, 11, 20]. However, the workload and the cache behavior of the stateful configuration have not been analyzed yet.

3 Environment and Methodology

3.1 Benchmarks Selection

According to the data management along the packet processing, there are the following application categories: *non-additional-data* applications are those programs that do not need to search any kind of data related to the packet or connection to be able to perform the packet processing. For example, CRC only needs the IP packet header. The *stateless* category includes the applications that generate no record of previous packet processing and each packet processing has to be handled based entirely on its own information. For instance, the packet forwarding do not record information about previous packet forwarding. Unlike non-additional-data category, stateless applications search information related to the packet (e.g. IP lookup data structures). Finally, the third category is the *stateful* applications that keep track of the state of packet processing [13], usually by setting fields of state related to the flows or connections. For example, TCP termination requires to maintain the state of the TCP flows. The main difference between stateful and stateless programs is the former may update a variety of fields within the state. Instead stateless applications only require the value and do not update any information. As our concern is to characterize stateful applications, the results of the other benchmarks are used to explain in a better way the differences among the above categories.

Table 1 shows the selected benchmarks according to the above classification. We select the benchmarks that present a similar performance than the average of the category. Due to the lack of stateful applications within the publicly available benchmark suites, there is only a single application that presents a potentially statefulness feature: Snort 2.3 [1]. We employ three different configurations: *Snort_SLess* is configured to execute stateless preprocessors, and

Table 1. Selected Benchmarks

App. Category	State Categories	Benchmark	Benchmark Suite
Non-Additional-Data	Pkt	AES	NpBench
	Pkt	MD5	NpBench
Stateless	Pkt & Global	Route	NetBench
	Pkt & Global	Nat	NetBench
	Pkt & Global	Snort_SLess	NetBench
Stateful	Pkt & Global & Flow	Snort_Str4	NetBench
	Pkt & Global & Flow	Snort_Pscan	NetBench
	Pkt & Global & Flow & App	Argus	

Snort_Str4 and *Snort_Pscan* are tuned to use the stateful preprocessors called Stream4 and Flow Portscan, respectively. The former is the inspection of establishing TCP connections and their maintenance and prevents attacks such as Snot [5] and Stick [2]. The latter is an engine designed to detect portscans based on flow creation and the goal is to catch one to many hosts and one to many ports scans. Additionally, we select *Argus* [4] (i.e. network Audit Record Generation and Utilization System) even it is not included in any benchmark suite. This application is a fixed-model Real Time Flow Monitor. That is, it can be used to monitor individual end systems or activity on the entire enterprise network.

The stateful granularity level of every application is shown in the column called "State Categories". We discuss and analyze this classification in Section 4.2.

3.2 Traffic Traces

In order to perform a strict comparison among applications from different benchmark suites we cannot use the default traffic traces included in the suites. Obtaining representative network traffic traces always has been an obstacle to overcome. There are several public sites (e.g. NLANR [16], CAIDA [6]) where there are publicly available traffic traces from a wide open range of routers (e.g. MRA, etc.). However, for confidentiality reasons the IP packet addresses of these traces are sanitized [18]. The sanitization of addresses involves the loss of spatial locality of the Internet IP address distribution [10] and it could affect the results of some networking application studies. Verdú et al. [23] shows that the loss of spatial IP address distribution has no significant influence on the evaluation of Snort with a stateful configuration. In fact, our analysis show that stateful applications do not present significant variations. This means that sanitized traffic is representative to do research in stateful applications. The non-additional-data applications are unaffected by the sanitization, because they do not need the IP address to perform any search. Finally, the stateless applications are affected by the use of sanitized traces. However, our studies show that they present no

Table 2. Baseline Configuration

Processor Configuration	
Fetch Width	4
Queues Entries	64 int, 64 fp, 64 ld/st
Execution Units	6 int, 3 fp, 4 ld/st
Physical Registers	192 int, 192 fp
ROB Size	256 entries
Branch Predictor Configuration	
Branch Predictor Perceptron	256 perceptrons, 4096 x 14 bit local 40 bit global history
Branch Target Buffer	256, 4-way
RAS	32 entries
Memory Configuration	
ICache DCache	64KB, 2-way, 8 banks, 32B lines, 1 cycle access
L2 Cache	2MB, 8-way, 16 banks, 32B lines, 20 cycles access
Main Memory	500 cycles access
TLB	48-entry I + 128-entry D
TLB miss penalty	160 cycles

significant variations in the application performance and, moreover, currently there is no better way to perform this analysis.

In order to keep track of TCP connections, the traces have to hold packets in the two directions of the flows (i.e. packets from server to client and vice versa). In other case, the connection does not follow the TCP protocol and the flow state could not be updated. There is a reduced number of public traces with this property. We select traces from an OC12c (622 Mbit/s) PoS link connecting the Merit premises in East Lansing to Internet2/Abilene (i.e. MRA traces within the NLANR site).

Finally, there is no public traces with representative traffic aggregation levels with bidirectional traffic. As we need a trace with a representative traffic aggregation level in order to obtain representative results [24], we synthetically generate traffic traces simulating different bandwidths. From four original traffic traces of the same link we sanitized them using four mechanisms that assure the independence of IP addresses between traces. The traffic trace used simulates to a bandwidth link of roughly 1Gbps, showing 170K active flows on average.

3.3 Evaluation Methodology

We use different analysis tools depending on the target of the study. We instrument the binary code with ATOM [21] and generate statistics for instruction distribution. On the other hand, for studying the performance of selected ap-

plications, we use a modified version of the SMTSim simulator [22]. Table 2 shows the configuration employed of a single threaded out-of-order processor. The baseline is an ample configuration in order to understand the actual application behaviour.

We run every benchmark using the selected traffic traces and processing the same number of packets. Before starting to take statistics, we run the applications until the initial stage is finished, such as the creating of IP lookup table. Subsequently, the applications are warmed running enough packets in order to reach the stable behaviour of the program. Our studies indicate that 10K packets are enough. Also, these studies indicate that on average 50K packets is a representative amount of packets for obtaining representative statistics.

4 Characterization of Benchmarks

4.1 Instruction Mix & ILP

Our results show that roughly 50% of instructions are arithmetic, shift and logic operations. The great part of the applications present a similar percentage of branch operations (12% on the average). Only *AES* shows a lower percentage. Finally, an average of 40% of instructions are memory accesses. Moreover, roughly two thirds of the memory accesses are loads.

The instruction distribution shows minor variations according to the application categories. Unlike the non-additional-data applications, the other networking programs require the search of additional data in order to do the packet processing, such as IP lookup or flow state. Due to the search of data structures, stateless and stateful applications are slightly more memory stressed.

On the other hand, we evaluate the instruction level parallelism (ILP) of the applications. The processor configuration presents variations over some of the baseline parameters (see Table 2) towards avoiding any additional performance constraint. There are no limitations on both fetch bandwidth and functional units. The new configuration also presents an oracle branch predictor and a perfect memory system, where every memory access has one cycle latency.

Figure 1 shows the available ILP within the applications. We can observe that the available ILP is independent of the application category, although it is inherent to the application itself. For example, *MD5* presents an ILP of 2,3 against the 4,5 of *AES*, even though both of them belongs to the same category. This difference is emphasized with the average ILP of *Control Plane* and *Data Plane* applications of the NpBench benchmark suite, where the greatest part of the applications belongs to the non-additional-data category.

The evaluated stateful applications present an ILP of 3,7 on average. Actually, the data flow of the standard stateful packet processing present a reduced ILP. Since the type of packet processing cannot exploit the ILP like the other applications.

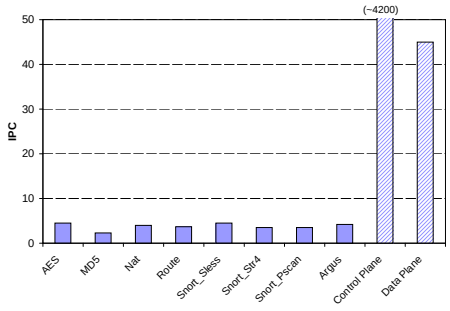


Fig. 1. Available Parallelism

4.2 State Categories

The variables and data accesses of an application are allocated into three segments of the memory model, namely: data segment, heap, and stack. From the application point of view, the data can be classified according to the variable environment, such as global to the application or local to a given function.

In general, the networking applications use to handle a global state. That is, the data structures are shared between packets. Actually, the data can be classified in two subsets. One is the global variables set, such as total number of processed packets. These variables are independent of any network traffic parameter. The other is represented by the global state data structures, such as IP lookup table. This state use to be related to a network feature, such as IP address. Thus, the temporal locality of these data between packets is determined by the network traffic properties. Generally, the lifetime of these data structures is eternal.

Nevertheless, the networking applications shows an additional classification that categorizes the data according to the granularity of the state that they represent:

- **Packet:** The data only cover the current packet processing, such as packet content. The lifetime of these data structures is very short, since they are reset every packet processing. The inter-packet temporal locality of this data set is independent of any network traffic parameter. In fact, depending on the target of the application, the temporal locality within the packet processing use to be very high.
- **Flow:** The data structures are related to the network connection, such as flow state. The temporal locality is determined by the traffic aggregation level of the network link. Although the lifetime is longer than Packet State Data, it is delimited by the lifetime of the flow.
- **Application:** The data structures are associated to the application layer, such as Real Time Protocol (i.e. a transport protocol designed to provide end-to-end delivery services for data with real-time characteristics). A similar granularity state level could be defined as macro-flow, such as counters of a

set of flows, which is used in monitoring/billing applications between others. The temporal locality is determined by the traffic aggregation level. However, the lifetime of the data is determined by the group of flows.

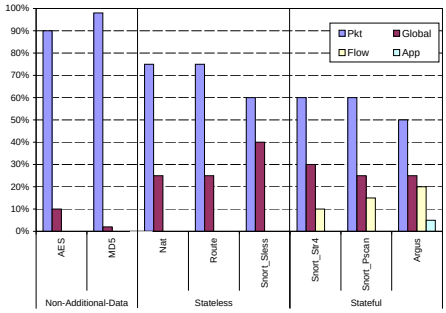


Fig. 2. Data Access Distribution

Lower statefulness granularity involves longer lifetime of the states and, in general, larger data structures. In fact, there can be more stateful data categories depending on the statefulness granularity of the application, namely: User State Data, which maintain state for every user; Department State Data, which maintain information for a group of users; etc.

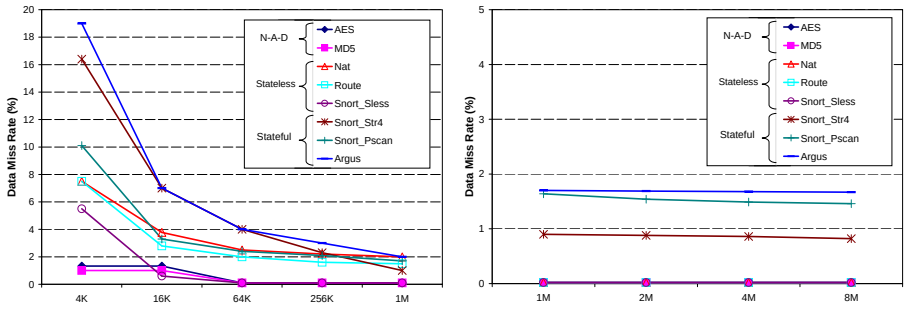
Figure 2 depicts the data access distribution depending on the data categories of the evaluated applications. The non-additional-data applications do a very high rate of data accesses on the Packet State Data. On the other hand, stateless applications use to search data for the packet processing. Due to this, the rate of Packet Data accesses is lower, whereas the Global Data access rate is higher. Depending on the target of the application, this search could be intensive, such as rule matching in a rule-based Intrusion Detection System. Finally, stateful applications present access rates to a wide variety of state data structures. The rate of flow state accesses can vary according to the statefulness granularity level of the application. Due to the properties of every data category, the data access distribution of an application directly affects the data cache behaviour.

4.3 Data Cache Behavior

In this section we discuss the data cache miss rate of the evaluated applications, and we analyze the impact of the data access distribution shown in the above section. We do not include an analysis of instruction cache. As other papers explain [25, 14, 12], the networking applications present on average near to 100% of instruction cache hit rates. Moreover, our studies show similar results with stateful applications.

Figure 3(a) shows the L1 data cache miss rate using a variety of sizes. The x-axis shows the sizes of L1 data cache. In the legend of the graph we group

the benchmarks according to the application category. We can observe that the non-additional-data applications present very reduced data cache miss rate, even with reduced size data caches. The greatest part of the working set is related to the packet state. Thus, the data cache is very useful in order to maintain this type of structures. The working set of stateless applications is larger due to global data structures, such as IP lookup table. As the access rates to these structures is higher, reduced caches present higher miss rates. However, a cache size of 16 KBytes or higher presents a low data cache miss rate of 2% on average. On the other hand, *Snort_SLess* could present higher miss rates if it was a rule-based IDS and there was a large number or rules. Finally, stateful applications present higher miss rates than the rest of applications, due to a larger working set and the variety of data structures that are dependent on networking properties, such as distance between two packets of the same flow.



(a) L1 Data Cache Miss Rate (b) L2 Data Cache Miss Rate

Fig. 3. Data Cache behaviour

We analyze the effects of increasing the associativity level, but there are no significant benefits. Because the main problem are not cache conflicts, but the cache inability of maintaining the data of the active flows. However, using a larger cache line size we take advantage of spatial locality within the packet processing itself. For example, if we enlarge from 32B up to 64B cache line size we achieve a 3% reduction on average in the cache miss rate. This fact represents three times the improvement obtain doubling the cache associativity degree.

Nevertheless, the most important impact resides in the L2 data cache miss rate shown in Figure 3(b). Whereas non-additional-data and stateless applications present near to zero miss rate, stateful applications show a saturated miss rate from 1% up to 1,7%. In other words, even using a large L2 data cache, the misses due to stateful data structures cannot be eliminated. Thus, when we use a very large DL1 cache, almost 100% of DL1 misses also are misses in L2.

Depending on the size of flow-states and the traffic aggregation level, we can require several MBytes of data structures for flow-states. In fact, the memory performance of stateful applications is very sensitive to the traffic aggregation

level [24], since the memory capacity requirements significantly grows and the temporal locality of flow state is reduced. Obviously, with larger flow-states the sensitivity is higher. Our studies show that *Snort_Str4* and Argus requires roughly 420 Bytes and 1 KByte, respectively. As we mentioned in Section 3.2, we maintain 170K flows on average. Thus, the memory requirements of *Snort_Str4* and Argus are roughly 68 MBytes and 166 MBytes, respectively. Therefore, this requirement will be higher with more statefulness applications that process traffic with higher aggregation level.

4.4 Branch Prediction

In this section we evaluate the behaviour of the branches through the study of branch prediction accuracy. The impact of branches on the IPC will be discussed in Section 4.5. We employ a perceptron predictor [26, 27]. Additionally, we also have studied other branch predictors, such as g-share [28]. However, the results show a slightly lower accuracy than the perceptron predictor, and higher sensitivity to the PHT size.

As we can see in Section 4.1, all the benchmarks present between 10% and 20% of branches within the workload, excepting *AES* that shows a lower rate. On average 70% of these branches are conditional branches.

The results show a high branch prediction accuracy for every benchmark. In fact, *AES*, *MD5*, and *Snort_SLess* present more than 99% of accuracy. *Snort_Pscan* and *Argus* shows roughly 98% of hit rate. Finally, the lowest accuracy rate is 95% by *Snort_Str4*. Our studies show that, as well as this application presents high branch rate, the branches are more sensitive to the network properties than the other applications. Thus, there is a negative aliasing among independent packets, unlike *Snort_Pscan* and *Argus* that present lower negative aliasign.

The key insight is that certain branches depend on the flow-state itself. Thus, regarding the statefulness of the application, coupled with the traffic aggregation level, the likelihood of obtaining lower branch prediction accuracy could be stronger.

4.5 Impact of Bottlenecks

Figure 4 presents the IPC of every benchmark using four different configurations: the baseline configuration, an oracle branch predictor, a perfect memory system (i.e. every memory access has one cycle latency), and a oracle predictor with a perfect memory system.

The Stateless applications present a baseline IPC of roughly 2,75. The main improvement resides in the branch prediction with a 12% on average. With the perfect configuration of memory and branch prediction we achieve an average of 20%. However, as *Snort_SLess* shows lower miss rates, its preformance improvement with the perfect configuration is only 2%. The main problem is the global data structure properties, such as size and locality. On the other hand, the stateful applications emphasize a lot the properties of global data structures through the accesses to flow and application data structures. Due to this, the memory

bottleneck is stressed and L2 data cache is unable to maintain the states of the active flows. A perfect memory system can obtain roughly 3x of speedup on average. The baseline IPC could be lower and the speedup could be emphasized when statefulness of the application is higher. Once the memory bottleneck is overcome, we can achieve an additional improvement of 12% on average.

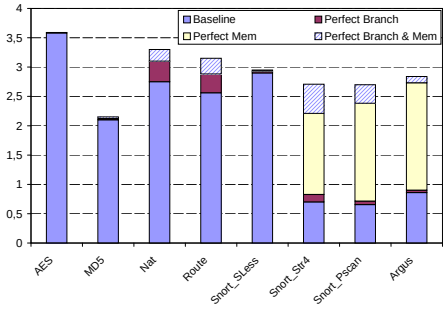


Fig. 4. Impact on IPC

5 Conclusions

To the best of our knowledge, we present the first workload characterization of stateful networking applications. We analyze the main differences between the networking application categories according to the management of state data. Actually, stateful applications present a variety of statefulness granularity levels.

The main bottleneck of stateful applications resides in the memory system, since even L2 data cache is unable to maintain the state of active flows. The evaluated applications show 3x of speedup on average using a perfect memory system. Nevertheless, this speedup can vary depending on the statefulness level of the application.

As future work we are concerned about to do a deep analysis of the architectural impact of the stateful applications. On the other hand, as the variety of stateful benchmarks is still reduced, we continue looking for other publicly available stateful applications.

Acknowledgements

This work has been supported by the Ministry of Education of Spain under contract TIN–2004–07739–C02–01, and grant BES-2002-2660 (J. Verdú), the HiPEAC European Network of Excellence, and the Barcelona Supercomputing Center. The authors also would like to thank Rodolfo Milito for his review and technical inputs of this work.

References

1. J. Beale, J. C. Foster, J. Posluns, and B. Caswell. *Snort 2.0 Intrusion Detection*. Syngress Publishing Inc., 2003.
2. G. Coretez Fun with Packets: Designing a Stick. Draft White Paper on Stick. *http://www.eurocompton.net/stick/*.
3. Cisco IOS NetFlow. *http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml*.
4. Argus - Auditing Network Activity. *http://www.qosient.com/argus*.
5. Snot V0.92 alpha. *http://www.stolenshoes.net/sniph/snot-0.92a-README.txt*.
6. Cooperative association for internet data analysis. *www.caida.org*.
7. The Computer Emergency Response Team. *www.cert.org*.
8. The System Administration, Networking and Security Organization. *www.sans.org*.
9. P. Chandra, F. Hady, R. Yavatkar, T. Bock, M. Cabot, and P. Mathew. Benchmarking network processors. In *Proc. NP1, Held in conjunction with HPCA-8*, Cambridge, MA, USA, Feb. 2002.
10. E. Kohler, J. Li, V. Paxson, and S. Shenker. Observed structure of addresses in IP traffic. In *Proc. of the 2nd ACM SIGCOMM Workshop on Internet measurment workshop*, Pittsburgh, PA, USA, August 2002.
11. C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In *Proc. IEEE Symposium Security and Privacy, IEEE Computer Society Press*, CA, USA, 2002.
12. B. K. Lee and L. K. John. Npbench: A benchmark suite for control plane and data plane applications for network processors. In *Proc. of ICCD*, San Jose, CA, USA, Oct. 2003.
13. S. Melvin, M. Nemirovsky, E. Musoll, J. Huynh, R. Milito, H. Urdaneta, and K. Saraf. A massively multithreaded packet processor. In *Proc. of NP2, Held in conjunction with HPCA-9*, Anaheim, CA, USA, Feb. 2003.
14. G. Memik, W. H. Mangione-Smith, and W. Hu. Netbench: A benchmarking suite for network processors. In *Proc. of ICCAD*, San Jose, CA, USA, Nov. 2001.
15. A. Nemirovsky. Towards characterizing network processors: Needs and challenges. Nov. 2000. Xstream Logic Inc., white paper.
16. National lab of applied network research. *http://pma.nlanr.net/Traces*.
17. A. M. Odlyzko. Internet traffic growth: Sources and implications. In *Optical Transmission Systems and Equipment for WDM Networking II, B. B. Dingel, W. Weiershausen, A. K. Dutta, and K.-I. Sato, Proc. SPIE, vol. 5247*, Sept. 2003.
18. R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proc. of the ACM SIGCOMM Conference*, Germany, Aug. 2003.
19. M. Roesch. Snort lightweight intrusion detection for networks. In *Proc. of the 13th Conference on Systems Administration (LISA)*, Seattle, WA, USA, Nov. 1999.
20. L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In *Proc. of RAID-6*, Pittsburgh, PA, USA, Sep. 2003.
21. A. Srivastava and A. Eustace. ATOM - A system for building customized program analysis tools. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 196–205, June 1994.
22. Dean M. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, Dec. 1996.

23. J. Verdú, J. García, M. Nemirovsky, and M. Valero. Analysis of traffic traces for stateful applications. In *Proc. of NP3, Held in conjunction with HPCA-10*, Madrid, Spain, Feb. 2004.
24. J. Verdú, J. García, M. Nemirovsky, and M. Valero. The Impact of Traffic Aggregation on the Memory Performance of Networking Applications. In *Proc. of MEDEA Workshop, Held in conjunction with PACT-2004*, France, Sept. 2004.
25. T. Wolf and Mark A. Franklin. Commbench - a telecommunications benchmark for network processors. In *Proc. of ISPASS*, Austin, TX, USA, April 2000.
26. D. Jimenez and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems 20(4):369-397*, Nov. 2002.
27. L. Vintan and M. Iridon. Towards a high performance neural branch predictor. In *Proc. of IJCNN*, vol. 2, pag. 868–873, July 1999.
28. Scott McFarling. Combining Branch Predictors. Technical Report TN-36, Compaq Western Research Lab, June 1993.