Architectural Impact of Stateful Networking Applications

Javier Verdú
DAC, UPC
Barcelona, Spain
jverdu@ac.upc.edu

Jorge García
DAC, UPC
Barcelona, Spain
jorge@ac.upc.edu

Mario Nemirovsky
Consentry Networks Inc.
Milpitas, CA, USA
mario@consentry.com

Mateo Valero
DAC, UPC
Barcelona Supercomputing Center
Barcelona, Spain
mateo@ac.upc.edu

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 grew both in traffic volume, and in the richness of the applications it supports. The growth also brought along new security issues and network monitoring applications. Edge devices, in particular, tend to perform upper layer packet processing. A whole new class of applications require stateful processing.

In this paper we study the impact of stateful networking applications on architectural bottlenecks. The analysis covers applications with a variety of statefulness levels. The study emphasizes the data cache behaviour. Nevertheless, we also discuss other issues, such as branch prediction and ILP. Additionally, we analyze the architectural impact through the TCP connection life. Our results show an important memory bottleneck due to maintaining the states. Moreover, depending on the target of the application, the memory bottleneck may be concentrated within a set of packets or distributed along the TCP connection lifetime.

1. INTRODUCTION

The explosive and robust growth of the Internet owes a lot to the "end-to-end principle" [6], which 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. For instance, in 2001 a worm called Code Red infected over 350,000 hosts over a week and the infection rate was doubling in about 37 minutes. Two years later, the Sapphire/Slammer worm infected more than 90 percent of vulnerable hosts in the world within only 10 minutes [16].

The attackers take advantage of stateless firewalls that cannot look beyond the single individual packet while inspecting network traffic. The attacks may be spread over 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 [10, 11, 12]. Moreover, stateless Network Intrusion Detection Systems (NIDSs) may be overwhelmed with other attacks, such as Snot [2] and Stick [3]. These attacks work by generating packets that are expected to trigger alerts by the NIDS, seeking to cause so many alerts that valid attack traffic will be able to slip by unnoticed in the noise. This heavy volume of alerting can even cause a NIDS crashes under the load of wanalyzing each packet and producing the corresponding alerts [4, 5]. Therefore, more complex firewalls, that keep track of the processed packets, are being developed in order to catch these new attacks.

The statefulness can present different granularity levels. From flow level, such as number of packets of a given flow, up to user or higher level, such as where do the users go on the network. Therefore, audit and monitoring applications heighten system administrators' awareness of network usage practices. In fact, there are other stateful applications that show the same trend, such as billing, visualization, malware, anomaly detection, stateful worm detection, stateful packet inspection, etc.

On the other hand. the development of network processors (i.e. high performance, programmable devices designed to efficiently execute communication workloads [14]) 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 [24]. Therefore, the traffic aggregation level reflects an incremental trend and stateful applications manage more flow states. Consequently, the memory capacity requirements become a bottleneck.

Independently of the processor employed, memory bottlenecks are already a challenge in the area of networking applications. However, in the upper layer applications, and specially stateful applications, the effects of memory latency become even more significant [15]. Since they have to maintain larger data structures, such as flow-states, from a growing number of different flows, the memory requirements may be tens or hundreds of MBytes.

In this paper we study the impact of stateful networking applications on architectural bottlenecks. The analysis covers applications with a variety of statefulness levels. The study emphasizes the data cache behaviour. Nevertheless, we also discuss other issues, such as branch prediction and ILP. Additionally, we analyze the architectural impact through the TCP connection life. Our results show an important memory bottleneck due to maintaining the states of the applications. Moreover, depending on the target of the application, the memory bottleneck may be concentrated within a set of packets or distributed along the TCP connection lifetime.

The remainder of this paper is organized as follows. Section 2 provides related work on the evaluation of the different benchmark suites. In Section 3, we present the selection and main properties of the evaluated benchmarks and traffic traces, as well as the evaluation methodology. The architectural analysis is presented in Section 5. Finally, we conclude in Section 6.

2. RELATED WORK

Benchmarking NPs is complicated by a variety of factors [13] (e.g. several programming models and languages, wide variety of application domains, 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 [22].

Several benchmarks suites have been published in the NP area: CommBench [32], NetBench [21] and NpBench [19]. Wolf et al. [32] present a set of eight benchmarks called CommBench. It is focused on program kernels typical of traditional routers. The benchmarks are distributed within two groups according to whether they are Header Processing Applications (HPA) or Payload Processing Applications (PPA). In this publication, the workloads are characterized and compared versus SPEC benchmarks.

Memik et al. [21] present a set of nine benchmarks (although in the available suite there are ten) 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 characterization of the workload is compared versus MediaBench programs.

Lastly, Lee et al. [19] 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. As we can see in the above benchmark suites, there is a lack of stateful benchmarks, since they do not keep track of the previous processed packets.

The above benchmark suites only have a single stateful program, called Snort [1], which is included in the NetBench suite. There are several publications that present studies about Snort [26, 18, 27], although the statefulness impact and properties are not studied. Depending on Snort's configuration, the statefulness of the processing may vary a lot.

In this paper, we evaluate different stateful benchmarks using a variety of stateful configurations of Snort. In addition, we study a publicly available monitoring/billing benchmark.

3. ENVIRONMENT & METHODOLOGY

3.1 Benchmarks Selection

Due to the lack of stateful applications in the above publicly available benchmark suites, we can only use one benchmark that is included in them, which presents a potentially statefulness feature: Snort 2.3 [1].

Depending on the configuration of Snort, the behavior may be completely different. On the one hand, it can be configured as a completely stateless pattern matching rule based IDS. On the other hand, it can be tuned as a completely stateful IDS. In this paper we employ different configurations in order to obtain a variety of stateful applications. Every configuration shows different footprints, with the exception of the packet decoding that is carried out regardless of the Snort configuration. Nevertheless, it represents a reduced percentage of the packet processing and it is not significant for the results of the evaluation.

We use four different configurations, namely: *Stream4* keeps track of the TCP connection state and prevents the Stick/Snot [3, 2] attacks; *Flow–Portscan* detects portscans based off flow creation and the goal is to catch one to many hosts and one to many ports scans; *SfPortscan* detects portscans to be able to detect the different types of scans Nmap (i.e. the most common port scanning tool in use today) can produce; *Merged Engines* the combination of the above engines. The combination of engines do not necessarily involve the aggregation of the analysis results. Since there are engines that can share flow state and, therefore, they can achieve a more robust application.

Additionally, we select *Argus* [8] as an equivalent publicly available benchmark to the monitoring/billing commercial tool called Cisco NetFlow [7], even though it is not included in any benchmark suite. Argus 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. Argus monitors and reports the status and performance of all network transactions examined in the network data stream.

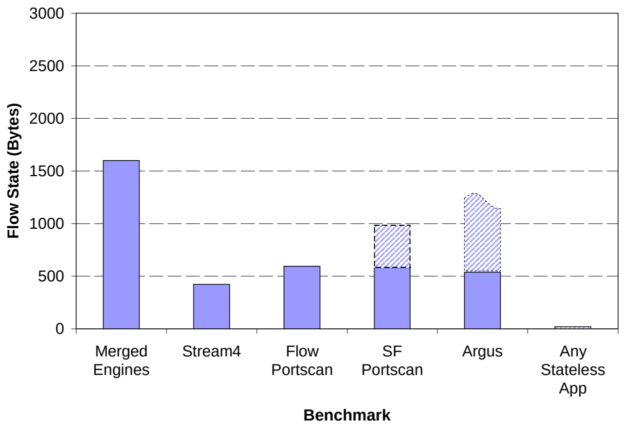


Figure 1: Statefulness of Benchmarks

Figure 1 depicts the flow state maintained by every application. Firstly, we can observe the contrast between any stateless program and the rest of stateful application. Merged Engines configuration shows the largest flow state with roughly 1,5 KByte. SfPortscan shows approximately 1

KByte of state, although some flows only requires roughly 600 Bytes. Argus present a flow state base of over 500 Bytes. However, depending on the configuration, the state can be quite more than 1KByte. We use a configuration that activates, at the most, 1KByte of state. The impact of state size and its management will be studied in subsequent sections.

3.2 Traffic Traces

Obtaining representative network traffic traces always has been an obstacle to overcome. There are several public sites (e.g. NLANR [23], CAIDA [9]) 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 [25]. The sanitization of addresses involves the loss of spatial locality of the Internet IP address distribution [17] and it could affect the results of some networking application studies. Verdú et al. [30] shows that the loss of spatial IP address distribution has no significant influence on the evaluation of stateful applications. This means that sanitized traffic is representative to do research in Snort.

As the stateful configuration of Snort maintains information of the history of TCP connections, traffic traces have to be surveyed. 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 (the packet processing would always generate an alert). Due to this, there is a reduced number of public traces useful for our studies. 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). Additionally, as the stateful benchmark is aimed to TCP packets, the remaining protocols (e.g. UDP, ICMP, etc.) do not generate any workload. Therefore, we filter the traffic traces and finally we obtain traffic traces with only TCP packets. Actually, the unique consequence of this filtering is the reduction of traffic bandwidth within the trace.

Finally, there is no traffic traces with representative traffic aggregation levels publicly available with bidirectional traffic. As we need a traffic trace with a representative traffic aggregation level in order to obtain representative results [31], 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. We establish the time stamp of the new trace from the packet time stamp of one of the traces. Then we mix the traces taking the packets sorted with the microsecond time stamp. Finally we obtain a new sanitized traffic trace that represents roughly four times wider bandwidth than the original link. The traffic trace used simulates to a bandwidth link of roughly 1Gbps, which presents 200K active flows on average.

3.3 Evaluation Methodology

In order to perform the analysis presented in this paper we use different tools depending on the target of the study. For analyzing the instruction distribution, we use ATOM [28]. We instrument the binary code and generate statistics both per packet and on average throughout the processed traffic trace. On the other hand, for studying the performance of selected applications, we use a modified version of the SMTSim simulator [29]. We simulate a single threaded out-of-order processor with the configuration shown in Table 1. The baseline is an ample configuration in order to be able to understand the application behaviour and to find the actual application bottlenecks.

Table 1: Baseline Configuration

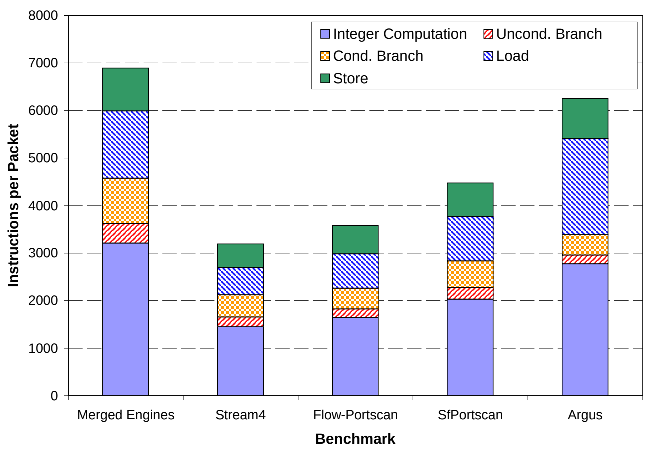
Processor Configuration	
Fetch Width	4
Queues Entries	64 int, 64 fp, 64 ld/st
Execution Units	6 int, 3 fp, 4 ld/st
Physical Registers	192 int, 192 fp
ROB Size	256 entries
Branch Predictor Configuration	
Branch Predictor Perceptron	256 perceptrons, 4096 x 14 bit local and 40 bit global history
Branch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

For obtaining comparable statistics, we need to set different simulation issues. 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 end, 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 for the warming stage of the applications. Also, these studies indicate that on average 50K packets is a representative amount of packets for obtaining stable statistics, since a larger number of processed packets does not show significant variations.

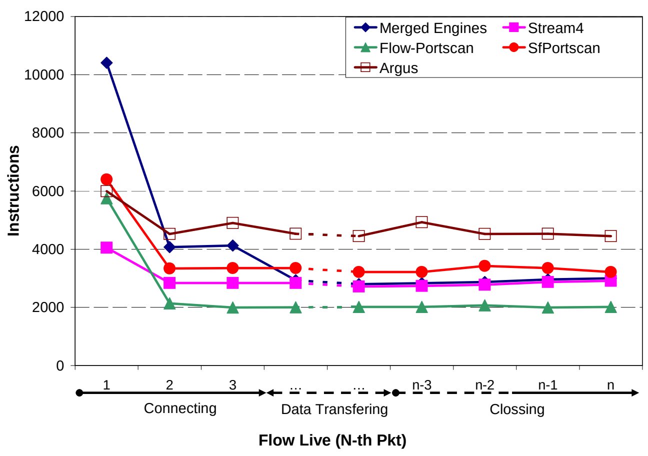
4. NETWORK TRAFFIC PROPERTIES

The performance of networking applications is closely related to the network traffic properties. According to the type of application the set of relevant network properties may vary. Stateless programs are prone to be sensitive to the IP address distribution along the network stream. Unlike stateful applications that are not sensitive to the IP address distribution [30]. For these applications there are other properties that affect directly the stateful workload performance, namely:

- **Traffic Aggregation Level** is a measurement of active flows. It indicates the rate of unique flows within a window of packets (e.g. unique flows within packets per second). The lowest level is given by a single active connection. Unlikely the highest level, where all packets are from a different active connection. Thus,



(a) Instruction Distribution



(b) Workload generated through the flow lifetime

Figure 2: Computational Complexity

higher traffic aggregation levels involve larger number of active states and, consequently, larger memory requirements.

- **Intra-Flow Temporal Distribution** shows how the packets are exchanged across the TCP connection lifetime, which is determined by the behaviour of the TCP window. Actually, the timestamp of the packets indicates the bursty nature of network traffic on the connection. Through the life of every TCP connection there are packet trains with different lengths. The packets within a packet train are closer than the other packets and, therefore, their packet processing is likelier to match the flow state in cache than the rest of packet processing.
- **Inter-Flow Temporal Distribution** indicates the distribution of the packets from all connections across the time. That is, the number of packets between two packets of the same flow. This property indirectly shows the traffic aggregation level of the link. Depending on this distribution, there can be a link that shows a large number of packets between two packets of the same flow, even these two packets are within a packet train of the connection that they belong to. Our studies on different public network traffic traces show that more than 50% of the traffic presents a distance between two packets of the same flow of hundreds and thousands of packets on low and high bandwidth links, respectively.

As we have explained throughout this section, the above properties have influence on the performance of stateful workloads. In fact, these properties show us the availability of taking advantage of shared features between packets, such as, flow state locality, footprint of packet processing of the same or different TCP connections, etc.

5. ARCHITECTURAL ANALYSIS

Throughout this section we present the architectural impact analysis of the stateful applications. The study is performed with both network traffic stream and every single packet through the TCP connection.

5.1 Instruction Mix and ILP

Figure 2(a) shows the computational cost of packet processing. We can observe that the relation among packet processing costs is alike the relation among flow-state sizes (see Figure 1). However, every benchmark shows similar instruction distribution rates. The greatest part of the processing is covered by roughly 45% of integer computation and 35% of memory accesses. The rest of the workload consists of roughly 12% and 8% of conditional and unconditional branches, respectively. Nevertheless, Argus is more memory intensive showing approximately 45% of memory accesses and less conditional and unconditional branches with roughly 7% and 3%, respectively.

The workload generated through the flow lifetime may be quite different according to the targets of the application. Figure 2(b) depicts the average of instructions executed by the n-th packet processing through the flow lifetime. The life of a connection presents three stages, namely: connecting, data transferring, and closing. The x-axis indicates the n-th packet. As the data transferring stage may be longer, it is represented with a dot line. The closing stage also may be delimited by different lengths. Our studies show that the instruction mix is similar in every packet processing, with the exception of the first packet of a TCP connection, which shows roughly 3% higher memory access rates, due to the initialization of values.

On the other hand, the Stream4 engine and Argus show a stable behaviour in every packet due to the constant task through the flow lifetime. However, when the other benchmarks are able to detect that the current flow is a safe connection, they leave the packet out of tracking the flow state. Therefore, there are two types of application according to the processing behavior along a networking connection, namely: the applications that the main stateful processing is concentrated on the first packets of the connection (e.g. portscanning); and the applications that spread the stateful processing along the packets of the TCP connection (e.g. monitoring), showing a similar workload in every packet.

In fact, we can see that once a connection is established Merged Engines generated lower workload than SfPortscan, although the former manages almost double flow state size than the latter. However, the former has engines that can share state and it involves that the application is more robust. Unlike the latter that requires to execute more instructions, because the engine is less "intelligent" due to the lack of additional state.

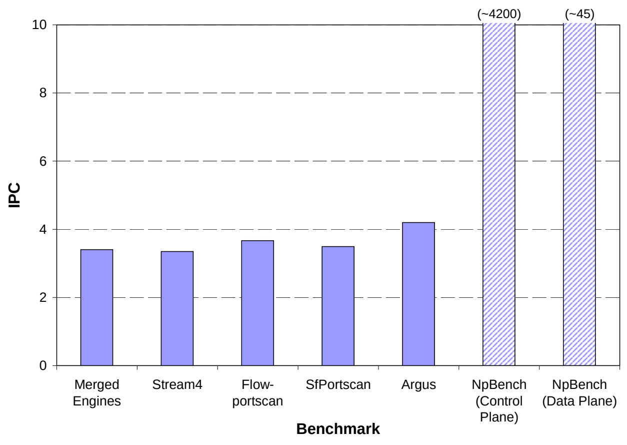


Figure 3: Available Parallelism

In order to explore the instruction level parallelism, the baseline configuration has been modified. The new configuration shows infinite fetch width, execution units, and reorder buffer entries, perfect memory system (i.e. 1 cycle access latency) and an oracle branch predictor. Figure 3 depicts the ILP of the evaluated stateful applications and the average of control and data plane benchmarks of NpBench suite [19]. Argus and the stateful configurations of Snort show a very reduced ILP (i.e. roughly 4 IPC) against the average of NpBench applications. Although stateful applications use to present a lot of dependencies through the packet processing, the ILP is inherent to the application itself as we can see in other networking application categories [19].

5.2 Performance Evaluation

In this section we assess the performance impact of branch miss predictions and cache misses. Figure 4 presents the IPC of every benchmark using four different processor configurations, namely: 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.

We can observe that Merged Engines shows a very reduced IPC of 0,12. The other benchmarks present higher IPCs, but all of them are lower than 0,9. Generally the relation among the IPC of the benchmarks should be similar to the relation among flow state size of the applications. However, we can observe that SfPortscan shows better performance than Flow-Portscan, even similar to Stream4. As we can see in the Figure 2(b), Stream4 presents a constant packet manegement, unlike the SfPortscan. Thus, the difference of the packet manegement involves a similar performance, even the flow state sizes are different.

The perfect branch configuration does not show significant improvements, unlike perfect memory configuration that can obtain an IPC speedup from 3x in Stream4 and Argus, up to 19x in Merged Engines. This means firstly that obviously the main bottleneck is the memory system. Secondly, once the memory bottleneck is solved, improving the branch prediction performance we still can obtain 16% of IPC speedup on average. And lastly, the speedup does not present a direct relation to the flow state size, but it is more related to the type of flow state manegement by the application.

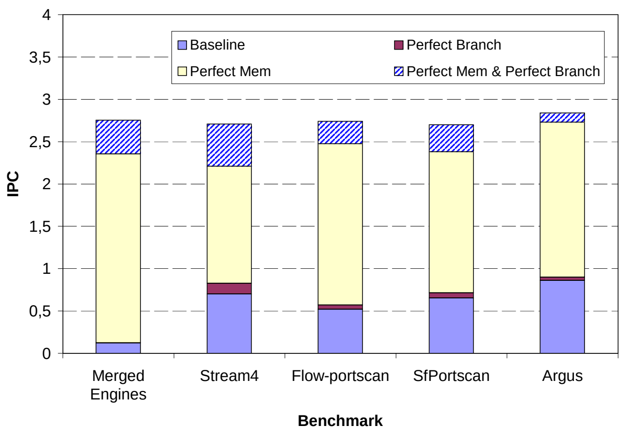


Figure 4: Impact on IPC

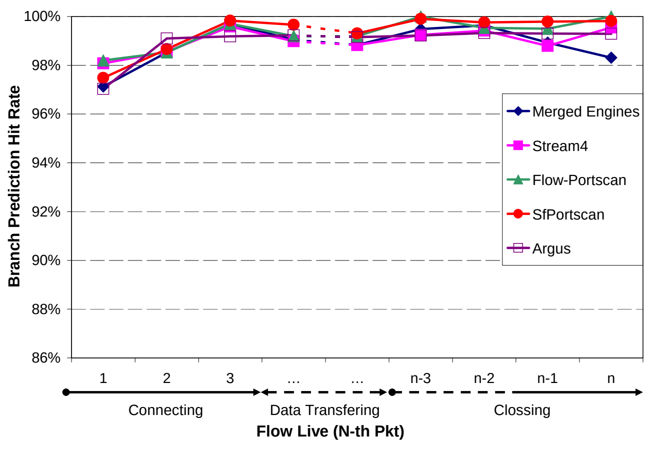
5.3 Branch Prediction

In this section we evaluate the behaviour of the branches through the study of branch prediction accuracy. We employ a perceptron predictor [33, 34]. We also have studied other branch predictors, such as g-share [35]. Nevertheless, the results show a slightly lower accuracy than the perceptron predictor, and higher sensitivity to the PHT size.

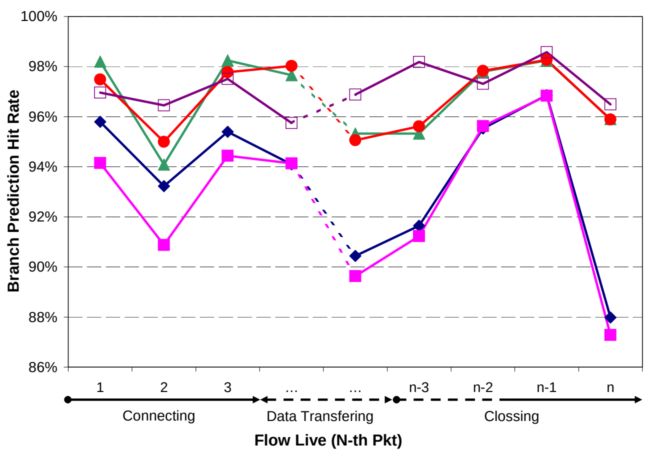
Figure 5 shows the branch prediction hit rate of every benchmark. On average there is a hit rate higher than 95%. We can observe that Stream4 is the benchmark with worse behaviour. On the contrary, Argus, Flow-Portscan, and SfPortscan present the highest hit rates. The results show that there is no relation between prediction hit rate and flowstate size (see Figure 1).

Before proceeding, we categorize the branches of the stateful applications, namely: flow independent and flow dependent branches. The former includes the branches that are executed for global tasks, such as updating global variables. These branches are insensitive to the previously mentioned traffic properties (see Section 4). Therefore, they are easy to predict, since the prediction is similar among packets. The latter includes the branches that update the flow state and variables related to the flow, such as state of the connection. As the prediction is dependent of the flow, these branches are sensitive to the network traffic properties. Due to this, packet processing of independent flows may cause negative aliasing in the branch prediction.

Figure 6(a) presents the branch prediction analysis across the flow lifetime with an environment of a single active connection. We can observe all benchmarks present a high branch prediction hit rate in every packet. In fact, the hit rate is higher than the average and there are no significant variations among the different n-th packet. On the other hand, Figure 6(b) shows the results across the flow lifetime with an environment of high aggregation traffic level. We can observe that, on average, there is a lower hit rate in every



(a) Branch Prediction - Traffic without aggregation level



(b) Branch Prediction - Traffic with high aggregation level

Figure 6: Branch prediction study through packet flow life

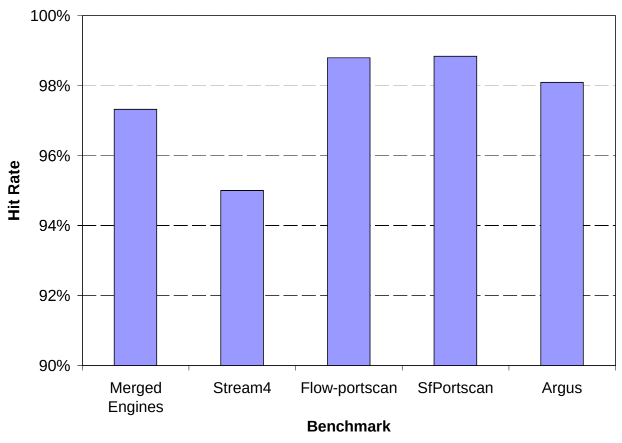


Figure 5: Branch Prediction Hit Rate

packet, specially Merged Engines and Stream4 present the most significant variations.

We can observe that there is negative aliasing due to the processing of packets from independent flows. In addition, there is a significant rate of flow dependent branches that are affected by the network traffic properties. Instead, the greatest part of the evaluated benchmarks concentrates the main workload in the first packets of the connection. Thus, the negative effect is hide, with the exception of Stream4 and Argus that present a similar workload throughout the n-th packets of the connection.

5.4 Data Cache Behavior

We do not include a study of the instruction cache. As other papers show [32, 21, 19], the networking applications present on average near to 100% of instruction cache hit rate. Nevertheless, we evaluate the selected applications of this paper. The results show that, using a single threaded processor, they have similar instruction cache performance than the other networking applications.

Figure 7 shows the data cache behaviour with a variety of sizes indicated by the x-axis. In the left graph we can see the DL1 miss rate. Increasing the cache size significantly reduces the miss rate with reduced caches. However, caches larger than 64 KBytes do not present important miss rate reductions. Merged Engines shows more than 5% of cache misses even using 1MByte.

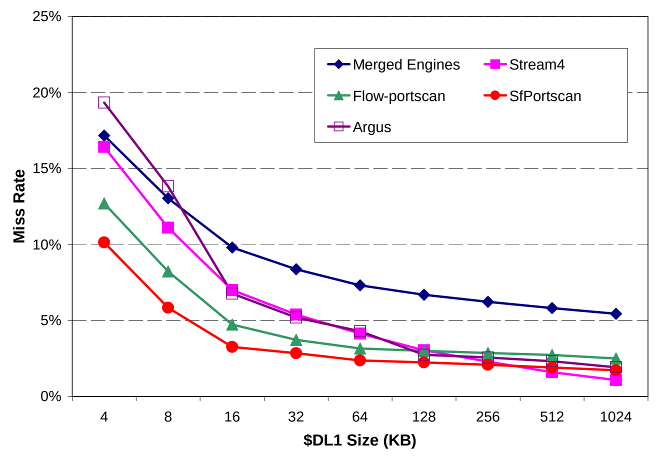
We can conclude that a DL1 cache of roughly 64 KBytes is able to maintain the variables that present temporal locality between packets, such as stateless data structures and global variables. Moreover, these data are insensitive to the network traffic properties of Section 4.

On the other hand, Figure 7(b) shows the L2 cache miss rate. We can observe an almost completely flat miss rate even with a very large L2 cache. Additionally, the L2 miss rates are almost the 100% of the miss rate of a very large DL1 cache. That is, roughly 5% of the memory accesses of Merged Engines are L2 cache misses, even using an 8 MBytes L2 cache. Therefore, even using several MBytes of cache, we are not able to reduce the miss rate generated by the stateful data.

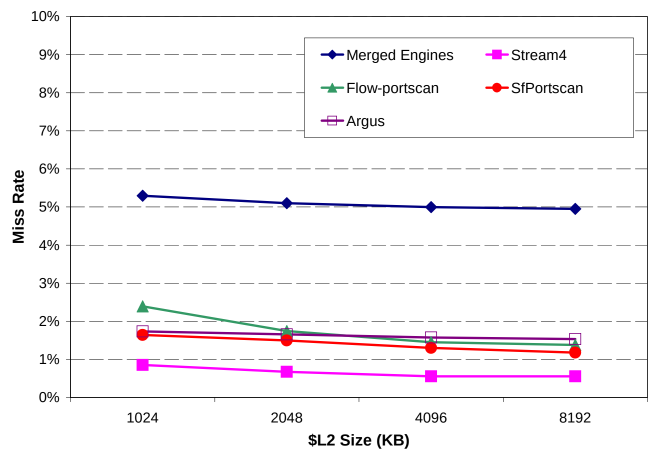
The network traffic properties have a stronger influence on the performance of cache when the flow states are larger. The applications that handle large flow states will get the saturated state even when the network link presents soft traffic properties. Instead, the intra–flow temporal distribution of the packets could also help to strengthen the state localities when there are packet trains.

Figure 8 shows the L2 miss rate generated by the stateful data accesses of every packet through the flow lifetime. The first packet processing of a connection shows the highest miss rate due to the initialization of the flow state. However, when the application is focused on detecting the safety of a TCP connection, such as Merged Engines, the negative memory effect is concentrated in the first packets. Unlike applications with constant workload, such as Argus and Stream4, that shows similar miss rate in every packet.

In addition, the support of stateful data structures generates additional accesses to independent stateful data, such as creating/deleting data structure nodes controlled by the application itself. Due to this, Figure 7(b) shows higher L2 miss rate than the average miss rate of the stateful data ac-



(a) $DL1 Miss Rate



(b) $L2 Miss Rate

Figure 7: Data Cache Behaviour

cesses of Figure 8. Merged Engines is the benchmark that presents the largest increase of L2 misses from 1,5% up to 5%. Unlike other benchmarks that show no significant extra miss rate, since the additional data accesses are lower and the likelihood of maintaining the extra data in cache is stronger.

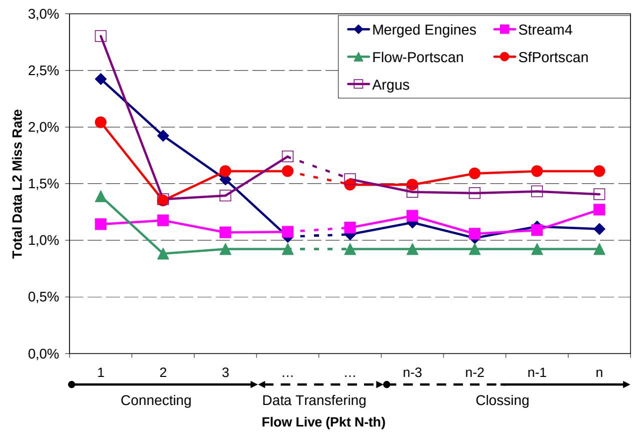


Figure 8: $L2 Cache Misses through Flow Life

6. CONCLUSIONS

This paper presents the first architectural impact analysis of a variety of stateful networking applications. We have explained how the performance and how the network traffic properties have influence on the performance of stateful applications, since the trend of Internet emphasizes the stress of the bottlenecks.

Moreover, we have shown that depending on the target of the application, the workload can be concentrated in the first packets of the TCP connection, such as port scan detector, or can be distributed along the flow lifetime, such as monitoring. This property is important for proposing state management optimizations or even architectural designs.

The main bottleneck is the data cache, and specially the L2 cache inability of maintaining the state of active flows. A perfect memory system can improve the performance from 3x up to 19x of speedup. Nevertheless, once the memory bottleneck is solved, the branch prediction can be improved showing 16% on average of speedup.

Finally, although other stateful applications may present different valuable results, the critical bottlenecks will be the same and they may even be more stressed. Nevertheless, our concern is to study the behavior of running stateful applications in a more realistic environment, that is, running simultaneously several applications.

7. 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.

8. REFERENCES

[1] J. Beale, J. C. Foster, J. Posluns, and B. Caswell. *Snort 2.0 Intrusion Detection.* Syngress Publishing Inc., 2003.

[2] Snot V0.92 alpha. *http://www.stolenshoes.net/sniph/snot-0.92a-README.txt.* June, 2004.

[3] G. Coretez Fun with Packets: Designing a Stick. Draft White Paper on Stick. *http://www.eurocompton.net/stick/.* June, 2004.

[4] D. Newman, J. Snyder, and R. Thayer. Crying Wolf: False Alarms Hide Attacks. *Network World.* June, 2002.

[5] M. Merideth, and P. Narasimhan. Elephant: Network Intrusion Detection Systems that Don't Forget. *Hawaii International Conference on System Sciences (HICSS-38).* Big Island, HI. January 2005.

[6] J. Saltzer, D. Reed, and D. Clark. End-To-End Arguments in System Design. In *ACM Transactions on Computer Systems*, 2(4):277-288, 1984.

[7] Cisco IOS NetFlow. *http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml.*

[8] Argus - Auditing Network Activity. *http://www.qosient.com/argus.*

[9] Cooperative association for internet data analysis. *www.caida.org.*

[10] The Computer Emergency Response Team. *www.cert.org.*

[11] The System Administration, Networking and Security Organization. *www.sans.org.*

[12] The DefCon Conference website. *www.defcon.org.*

[13] 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.

[14] P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk. *Network Processor Design: Issues and Practices*, vol. 1, chapter Network Processors: An Introduction to Design Issues. Morgan Kaufmann Publishers, USA, 2002.

[15] P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk. *Network Processor Design: Issues and Practices*, vol. 2, chapter Network Processors: Themes and Challenges. Morgan Kaufmann Publishers, USA, 2003.

[16] J. Won-Ki Hong. Invited talk: Internet traffic monitoring and analysis using ng-mon. In *Proc. of IEEE ICACT-6*, Republic of Korea, Feb. 2004.

[17] 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*, PA, USA, August 2002.

[18] 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.

[19] 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.

[20] 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.

[21] G. Memik, W. H. Mangione-Smith, and W. Hu. Netbench: A benchmarking suite for network processors. In *Proc. ICCAD*, CA, USA, Nov. 2001.

[22] A. Nemirovsky. Towards characterizing network processors: Needs and challenges. Nov. 2000. Xstream Logic Inc., white paper.

[23] National lab of applied network research. *http://pma.nlanr.net/Traces.*

[24] 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, eds., Proc. SPIE, vol. 5247*, Sept. 2003.

[25] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proceedings of the ACM SIGCOMM Conference*, Germany, August 2003.

[26] M. Roesch. Snort lightweight intrusion detection for networks. In *Proceedings of the 13th Conference on Systems Administration (LISA-99)*, Seattle, WA, USA, Nov. 1999.

[27] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In *Proc. of the 6th International Symposium on RAID*, PA, USA, 2003.

[28] 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*, June 1994.

[29] Dean M. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, pages 819–828, Dec. 1996.

[30] 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.

[31] 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*, Juan-les-Pins, France, Sept. 2004.

[32] T. Wolf and Mark A. Franklin. Commbench - a telecommunications benchmark for network processors. In *Proc. of ISPASS*, TX, USA, 2000.

[33] D. Jimenez and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems 20(4):369397*, Nov. 2002.

[34] L. Vintan and M. Iridon. Towards a high performance neural branch predictor. In *Proc. of the International Joint Conference on Neural Networks*, vol. 2, pag. 868873, July 1999.

[35] Scott McFarling. Combining Branch Predictors. Technical Report TN-36, Compaq Western Research Lab, June 1993.