A Case for Run-time Adaptation in Packet Processing Systems*

Ravi Kokku‡ Taylor L. Riché‡ Aaron Kunze† Jayaram Mudigonda‡ Jamie Jason † Harrick M. Vin‡
‡University of Texas at Austin †Intel Corporation
{rkoku, riche, jram, vin}@cs.utexas.edu {aaron.kunze, jamie.jason}@intel.com

Abstract: Most packet processing applications receive and process multiple types of packets. Today, the processors available within packet processing systems are allocated to packet types at design time. In this paper, we explore the benefits and challenges of adapting allocations of processors to packet types in packet processing systems. We demonstrate that, for all the applications and traces considered, run-time adaptation can reduce energy consumption by 70-80% and processor provisioning level by 40-50%. The adaptation benefits are maximized if processor allocations can be adapted at fine time-scales and if the total available processing power can be allocated to packet types in small granularities. We show that, of these two factors, allocating processing power to packet types in small granularity is more important—if the allocation granularity is large, then even a very fine adaptation time-scale yields meager benefits.

1 Introduction

Packet processing systems (PPS) are designed to process network packets efficiently. Over the past several years, the diversity and complexity of applications supported by PPS have increased dramatically. Examples of applications supported by PPS include Virtual Private Network (VPN), intrusion detection, content-based load balancing, and protocol gateways. Most of these packet processing applications involve multiple types of packets; applications are specified as graphs of functions and the specific sequence of functions invoked for a packet depends on the packet's type (determined based on the packet header and/or payload) [12, 16]. For example, a Secure Socket Layer (SSL) [10] application processes three packet types—setup packets (that create per-flow state in the PPS), outgoing packets (that involve encryption), and incoming packets (that require decryption). Researchers are also proposing to incorporate rich functions into network overlays [6, 14]. The emergence of such networks will further increase the diversity of packet types processed by a packet processing system.

Historically, packet processing systems were implemented using either fixed-function hardware or general-purpose processors. However, the increased demand for flexibility and performance has led to the emergence of packet processing system designs based on programmable, multi-core network processors (NPs). Multiple processor cores enable NPs to exploit the packet-level parallelism inherent in applications, and thereby achieve high packet processing throughput. Individual processor cores available in the NPs, however, are each configured with a very limited size instruction store (e.g., 4K instructions in Intel®'s IXP2800 network processor); the limited instruction store is often sufficient to hold code for processing individual packet types, but rarely enough to hold code for all packet types. This leads to software designs in which the responsibility for processing different packet types is partitioned among the processor cores—with each core specialized to perform one function [1].

Today, the allocation of processor cores to packet types is done statically at design time[1]. Further, to guarantee robustness to fluctuations in the arrival rate for different packet types, packet processing systems often provision sufficient number of processors to handle the expected *maximum* load for each packet type. The observed load for each packet type at any instant, however, is often substantially lower than the expected maximum load; further, the observed load can fluctuate significantly over time. In such settings, an adaptive run-time environment—that can change the allocation of processors to packet types at run-time—can conserve energy by reducing the power consumption of idle processors (e.g., by turning off processors or running them in low-power mode). Further, by multiplexing processors among different types of packets, an adaptive system can reduce the cumulative processor requirement (or provisioning level), and thereby reduce system cost. In this paper, we ask the following fundamental question: *how significant are the benefits of dynamically adapting the processors allocated to process different types of packets within a packet processing system?*

The problem of adapting processor allocations has received considerable attention in server clusters (e.g., data centers) [4, 5, 8] and other domains [2, 9, 11, 19]. However, the problem of adapting processor allocations in packet processing systems has remained virtually unexplored.

In this paper, we take the first step in exploring the benefits and challenges of adapting allocations of processors to packet types in packet processing systems. We consider six canonical packet processing applications and traces collected from multiple network points. We demonstrate that, for all the applications and traces considered, run-time adaptation can reduce energy consumption by 70-80% and processor provisioning level by 40-50%. The adaptation benefits are maximized if processor allocations—the mapping of packet types to processors—can be adapted at fine time-scales and if the total available processing power can be allocated to packet types

*This work is supported in part by Intel® and NSF ITR grant ANI-0326001.

[1]Today, most NPs support processor cores with instruction stores and not instruction caches (as provided in general-purpose processors). Hence, the specialization of processor cores is performed at system design/initialization. Introducing support for instruction caches in NP cores does allow on-demand loading of instructions into the caches. However, provisioning such instruction caches does not eliminate the need to specialize processor cores. This is because, each packet processing application can be thought of as a large *loop* that repeats for every packet; to ensure high packet processing throughput, the entire loop body must fit into the instruction cache.

Packet type	Cycles
Setup	20.913 million
Outgoing data (encryption)	325 * *x*
Incoming data (decryption)	325 * *x*

Table 1: Work-model for SSL. *x* is the packet size in bytes

in small granularities (or units). We show that, of these two factors, allocating processing power to packet types in small granularity is more important—if the allocation granularity is large, then even a very fine adaptation time-scale yields meager benefits.

Our results expose a challenging and rich area for future research. In particular, we argue that realizing the adaptation benefits requires the design of a *low-overhead, adaptive runtime environment* for packet processing systems.

The rest of the paper is organized as follows. We discuss our experimental methodology for measuring adaptation benefits in Section 2. Section 3 describes the results of our experiments. In Section 4, we discuss the implications of our findings; and finally, Section 5 summarizes our contributions.

2 Experimental Methodology

Our objective is to derive the benefits of adapting the allocation of processors to packet types at run-time in packet processing systems. To derive these benefits, we need to characterize the fluctuations in the processing requirements for different packet types in applications. We achieve this in three steps. First, we profile a set of canonical packet processing applications; we identify the different packet types involved in these applications and derive the computational requirements for processing a packet of each type. Second, we analyze packet traces collected from several sites in the Internet and derive the fluctuations in the rate of packet arrival for each packet type. Third, we combine computational profiles derived for each packet type with its arrival rate fluctuations to compute the fluctuations in the processing requirements. We utilize this information to derive adaptation benefits. In what follows, we describe each of these steps in detail.

2.1 Applications and Work Models

We consider six canonical packet processing applications: Secure Sockets Layer [10], Network Address Translation [7], IPv4/IPv6 Interoperability [21], TCP/IP header compression and decompression [15], IPv4 forwarding [13], and a 3G-wireless router (that supports IPv4/v6 interop along with header compression functionality).

For each application in this set, we identify its important packet types and profile the instruction cycles taken—referred to as the *work model*—to process a packet of each type using the Performance Counters Library [3] on a 930MHz, Intel® Pentium® III system. Our set of applications cover a spectrum of possibilities; it includes applications with a small to a large number of packet types, as well as applications with highly skewed to roughly uniform distributions for the time required to process packets of different types. For brevity, we discuss the work models for the two applications: SSL and the IPv4/v6 interop application. SSL involves only three packet types—setup (that create per-flow state in the PPS), outgoing data (that involve encryption), and incoming data (that require decryption). Setup takes orders of magnitude greater time to process than outgoing or incoming data (see Table 1). In the IPv4/v6 interop case, an input packet can either be an IPv4 or an IPv6 packet; further, the processing requirements varies depending on whether the packet is an ICMP, TCP, or a UDP packet. The processing time required for these paths are described in Table 2. Work models for the remaining applications are described in [17].

Pkt. type	Cycles	Pkt. type	Cycles
IPv4 ICMP	4317	IPv6 ICMP	1656
IPv4 TCP FTP	16950	IPv6 TCP FTP	9387
IPv4 TCP Rest	12541	IPv6 TCP Rest	2949
IPv4 UDP DNS	20430	IPv6 UDP DNS	9042
IPv4 UDP Rest	12346	IPv6 UDP Rest	2837

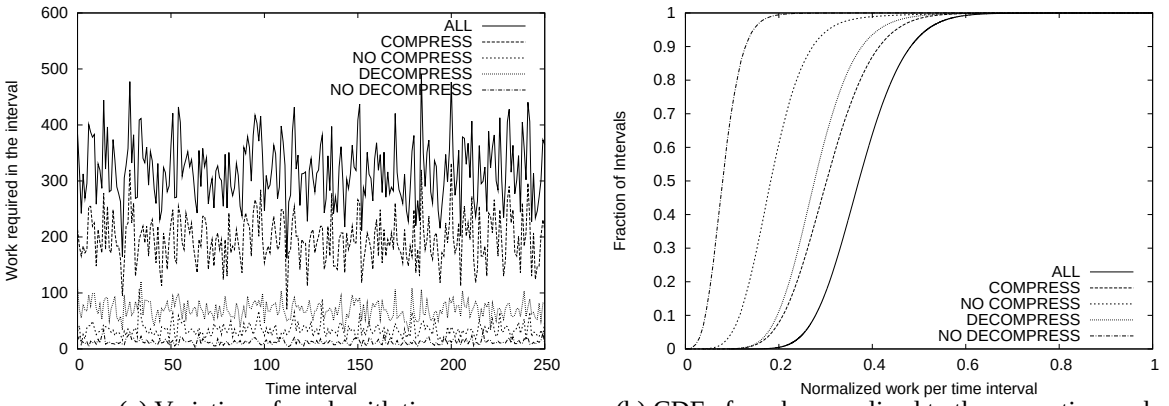
Table 2: Work-model for IPv4/IPv6 interoperability

2.2 Packet Traces

We analyze traces collected from various points in the Internet. For brevity, we only discuss two sets of traces—NLANR traces collected over 24 hours from a link connecting New Zealand to US [18], and one hour long traces collected from the high-speed link connecting University of North Carolina at Chapel Hill (UNC) to its network service provider [20]. These traces contain 60–100 million packets. We use these traces to estimate the fluctuations in the arrival rate for each packet type for the applications under consideration as follows.

Each of these raw traces consists of two packet sequences—incoming and outgoing. These two sequences represent packet arrivals of two different types. For instance, for the IPv4/IPv6 Interoperability application, all incoming packets are considered IPv4 packets and all outgoing are considered IPv6 packets. Similarly, for SSL, all packets that contain a TCP-SYN/SYN-ACK in either direction are marked as setup packets. All other incoming TCP packets are decrypted, and all other outgoing TCP packets are encrypted. Non-TCP packets are ignored. For the 3G-wireless router, all incoming packets are considered IPv4 packets, while all outgoing packets are considered IPv6 packets. Further, we assume that, with equal probability, each arriving packet has a compressed or an uncompressed header, each departing packet has a compressed or an uncompressed header, and each departing packet leaves as is or is translated to the other version of the protocol.

For each application, once the trace of packet arrivals is derived, the fluctuations in the processing requirement (or workload) for each packet type can be computed by scaling the fluctuations in the arrival rates for each packet types by the corresponding work model information. Figure 1(a) illustrates the variation in the processing requirements observed in the first 250 intervals (of length 5ms) of a UNC trace for different packet types of the TCP/IP header compression application. Figure 1(b) plots the cumulative distribution function (CDF) for the normalized workload (normalized with respect to the maximum observed workload during the entire trace) for each



(a) Variation of work with time (b) CDF of work normalized to the respective peaks

Figure 1: Analysis of the header compression application

packet type. These graphs show that the typical processing requirements can be significantly lower than the peak usage for all packet types, indicating that the benefits of run-time adaptation can be substantial.

2.3 Metrics of evaluation

We derive an estimate for the *ideal* benefits of adapting processor allocation at run-time in packet processing systems as follows.

We define τ as the smallest time interval for which the workload for each packet type is measured. Further, we define $\tau * I$, where I is an integer, as the length of the adaptation interval. In accordance with today's common practice (as discussed in Section 1), we assume that within any adaptation interval, processors allocated to a packet type can only process packets of that type; and the allocation of processors to packet types can only be changed at the boundaries of the intervals. Thus, the time duration $\tau * I$ defines the adaptation time-scale.

Let P be the set of packet types for an application, and let W_p^j denote the workload for some packet type $p \in P$ during the jth adaptation interval, among n adaptation intervals of size $\tau * I$. Note that W_p^j is equal to the number of packet arrivals of type p in interval j times the work model information for that packet type. The maximum workload for packet type p observed during any interval of size τ is given by: $\max_{j \in [1...n]} \left(W_p^j / I \right)$.

We define processor allocation granularity (or the unit of processor allocation) $\mathcal{G}$ in terms of the amount of workload a processor can process in time interval τ. In particular, we consider a range of values for $\mathcal{G}$ given by: $\mathcal{G} = g * W_{max}(\tau)$, where $0 < g \leq 1$ and $W_{max}(\tau) = \sum_{p \in P} \max_{j \in [1...n]} \left(W_p^j / I \right)$. Thus, in a system provisioned with sufficient processors to meet the maximum processing demands (given by $W_{max}(\tau)$), by selecting different values of g, we explore the spectrum where the total processing requirements are met by using a small number of large processors (i.e., with large values of $\mathcal{G}$) or a large number of small processors (i.e., small values of $\mathcal{G}$).

Given $\mathcal{G}$, the grunularity for processor allocations, the number of processors allocated to process packet type p in the jth adaptation interval (of length $\tau * I$) is given by $\left\lceil \frac{(W_p^j / I)}{\mathcal{G}} \right\rceil$.

To guarantee robustness to fluctuations in the arrival rate for different packet types, a non-adaptive system must provision sufficient processors to handle the *maximum* load for each packet type. Thus, the processor provisioning (denoted by $\mathsf{sum\_max}$) for a non-adaptive system is given by:

$$\mathsf{sum\_max} = \sum_{p \in P} \max_{j \in [1...n]} \left(\left\lceil \frac{(W_p^j / I)}{\mathcal{G}} \right\rceil \right) \tag{1}$$

The average number of processors (denoted by $\mathsf{sum\_avg}$) utilized in an adaptation interval is given by:

$$\mathsf{sum\_avg} = \frac{1}{n} \sum_{p \in P} \sum_{j=1}^{n} \left(\left\lceil \frac{(W_p^j / I)}{\mathcal{G}} \right\rceil \right) \tag{2}$$

An adaptive packet processing system can conserve energy by reducing the power consumption of idle processors within each adaptation interval. Thus, the adaptation benefits (denoted by $\mathsf{B_{total}}$) is equal to the average of the percentage of processors that are idle across all adaptation intervals, and is given by:

$$\mathsf{B_{total}} = 1 - \frac{\mathsf{sum\_avg}}{\mathsf{sum\_max}} \tag{3}$$

The maximum number of processors (denoted by $\mathsf{max\_sum}$) active during any adaptation intervals is given by:

$$\mathsf{max\_sum} = \max_{j \in [1...n]} \left(\sum_{p \in P} \left\lceil \frac{(W_p^j / I)}{\mathcal{G}} \right\rceil \right) \tag{4}$$

A system that can multiplex processors among different packet types would never utilize $(\mathsf{sum\_max} - \mathsf{max\_sum})$ processors. Thus, an adaptive system can help reduce the provisioning level of the system to $\mathsf{max\_sum}$; in doing so, it helps reduce the system cost. We denote this multiplexing benefit as $\mathsf{B_{mux}}$ and quantify it as follows:

$$\mathsf{B_{mux}} = 1 - \frac{\mathsf{max\_sum}}{\mathsf{sum\_max}} \tag{5}$$

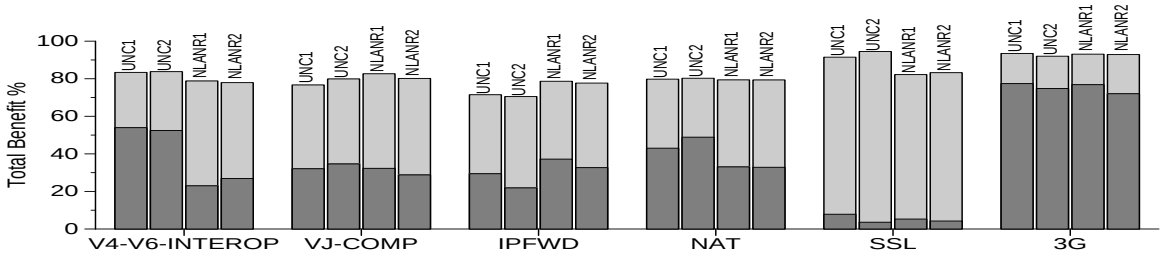


Figure 2: Benefits of adaptation for different applications and workloads. The dark grey bars represent B_{mux}.

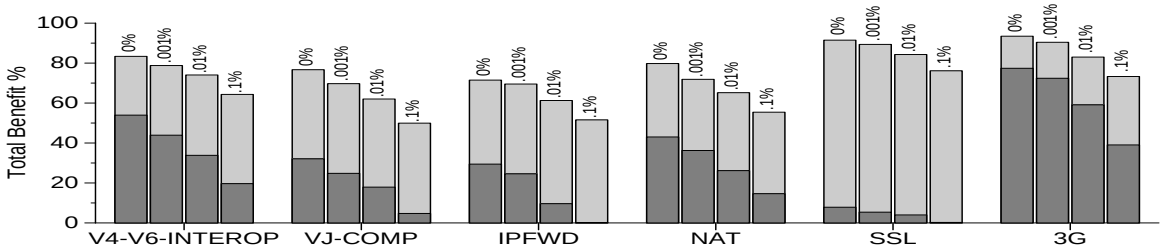


Figure 3: Benefits of adaptation for different applications with the UNC1 trace at processor provisioning levels.

To derive our estimate of the *ideal* adaptation benefits, we make several simplifying assumptions. First, we assume that processor allocation can be adapted instantaneously; the adaptation process itself imposes no overhead and consumes no processors. Second, we assume that an adaptive system allocates precisely the right amount of processors needed to process each packet type within each adaptation interval. Third, we assume that the processor requirements grow linearly with workload. Finally, we assume that the set of packets arriving within an interval are serviced by the end of the interval; and that the system contains sufficient memory to buffer packets for the duration of the adaptation interval. In reality, realizable benefits get influenced by many practical considerations, which we discuss in Section 4.

3 A Case for Run-time Adaptation

In this section, we analyze the benefits of adapting processor allocations to packet types as a function of three parameters: the processor provisioning level for a non-adaptive system (sum_max), the time-scale of adaptation (I), and processor allocation granularity ($\mathcal{G}$). For our analysis, we assume that τ, the minimum time-scale for measuring packet arrivals, is set to 100 times the average inter-packet arrival time in the trace (this translates to 4.7ms for the UNC trace).

Best-case Analysis: Because the workload for packet types fluctuate significantly at very small time-scales, adaptation benefits are maximized if processor allocations can be adapted at fine time-scales and if the processor allocation granularity can be small. To estimate this best-case benefits, we consider a system where processor allocation is adapted every τ time units (i.e., $I = 1$), the granularity of processor allocations is small ($g = 0.0001$), and the system is provisioned with sum_max processors. Figure 2 shows the total adaptation benefits B_{total} and the multiplexing benefits B_{mux} for all the applications and several traces. It illustrates that run-time adaptation can reduce energy consumption by 70-80% and processor requirements by up to 40-50% for all applications under all traces.

Figure 2 also illustrates that the contribution of B_{mux} to B_{total} varies across applications. This is because the workloads for the applications differ from each other along three dimensions: (1) number of packet types, (2) work model for the packet types, and (3) the correlation between the arrivals of different packet types. For example, in SSL, multiplexing benefits are significantly smaller than in other applications. This is because, SSL supports only three packet types and the work required to process setup packets is two orders of magnitude larger than that needed for other packet types (see Table 1). In this case, the workload within any interval is dominated by one packet type, thus reducing the opportunity for processor multiplexing. For the 3G-wireless application, on the other hand, multiplexing benefits are higher because of two reasons: (1) the application contains a large number of packet types and with similar computational requirements, and (2) the arrival rate of a particular packet type is not correlated to that of other types. This observation also indicates that a system that supports multiple independent packet processing applications will exhibit significant processor multiplexing benefits.

Variation with Processor Provisioning Levels: In this experiment, we evaluate the impact on the adaptation benefits of changing the processor provisioning level. In particular, we experiment with different provisioning levels such that in a non-adaptive system, at most L% (for different values of L) of the packets received within an adaptation interval cannot be serviced prior to the end of the interval (and hence experience significant queuing delays). Figure 3 illustrates that although the multiplexing benefits reduce with increase in L, the total adaptation benefits remain significant (50-70%) even for significant values of L (0.1%).

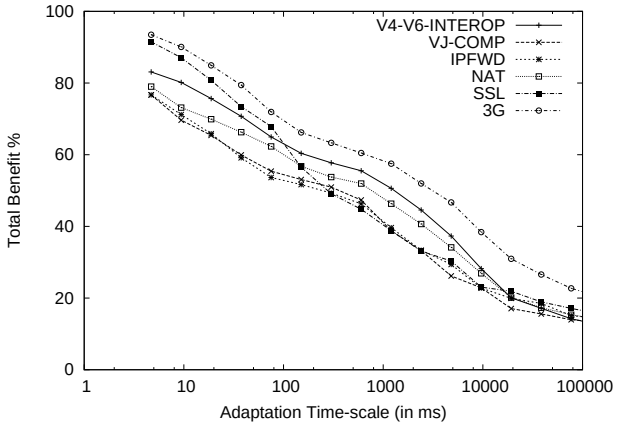


Figure 4: Variation of benefits with $\tau * I$ for various applications with the UNC1 trace.

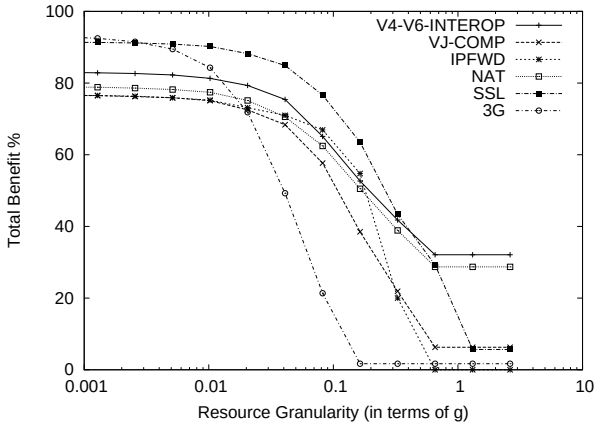


Figure 5: Variation of benefits with g for various applications with the UNC1 trace.

Variation with I and $\mathcal{G}$: We now evaluate the impact on adaptation benefits of varying the adaptation time-scale I and the processor allocation granularity $\mathcal{G}$ for different values of g; we consider the case where the system is provisioned with sum_max processors.

Figure 4 shows the effect of increasing I on the total adaptation benefits B_{total} (with $g = 0.0001$). It illustrates that the benefits decrease with increase in I. For instance, adapting processor allocations at 1 second time-scales yields 40-60% benefits (down from 80-90% for $I = 1$) for all the applications. The greater the adaptation time-scale (I), the greater the opportunity to average out instantaneous bursts of packet arrivals (and hence workload) over a longer duration, and hence the smaller is the difference between the required processor provisioning level (sum_max) and the average processor requirements. Observe, however, that provisioning processor resources to match the average requirements over longer time-scales increases the average delay incurred by packets.

Figure 5 shows the effect of increasing processor allocation granularity $\mathcal{G}$ (by increasing values of g) on the total benefits (with $I = 1$). It illustrates that when the average workload in an interval is sufficiently larger than the processor granularity $\mathcal{G}$, then increasing $\mathcal{G}$ has little effect on the adaptation benefits. However, as $\mathcal{G}$ approaches the average workload W_i^p of each packet type per interval, the adaptation benefits fall sharply.

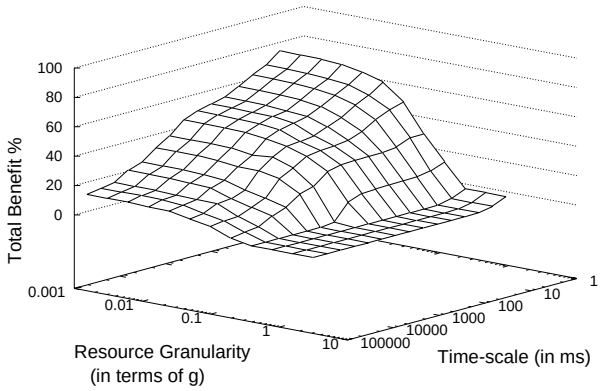


Figure 6: Variation of benefits with $\tau * I$ and g for the header compression application with the UNC1 trace.

The adaptation benefits become zero when the processor granularity is large enough ($\mathcal{G} \geq \max_{j \in [1...n]} W_i^p$) such that *at most* one processor per packet type is required in any interval. For some applications and traces, some of the intervals contain no packet of certain types; in these cases, an adaptive system can conserve energy by turning off those processors, and hence the benefit does not drop down to zero. This analysis suggests that an NP architecture that supports a large number of small processor allocation units will achieve higher adaptation benefits as compared to an architecture that supports a small number of large processor allocation units.

Figure 6 shows the effect of simultaneously varying both the time-scale of adaptation (I) and the granularity of processor allocation (g) on the total benefits. It illustrates that ensuring that allocating processors at small granularities is crucial. At coarse processor allocation granularities, regardless of the adaptation time-scales, the adaptation benefits are meager.

4 Run-time Adaptation: Challenges

In the previous section, we have demonstrated that run-time adaptation of processor allocation can yield significant benefits for packet processing systems. In this section, we argue that realizing these benefits can be quite challenging. It requires the design of a *run-time environment* for packet processing systems that can address the following fundamental questions: (1) when should processor allocations be adapted? (2) what should the revised processor allocation be? and (3) what mechanisms should the packet processing systems support to facilitate adaptation of processor allocations at small time-scales and processor granularities? Observe that the above questions need to be addressed in any system (e.g., server clusters in data centers) that supports dynamic adaptation of processor allocations. However, addressing these questions in packet processing systems is significantly more involved than conventional server cluster because of the following two reasons.

First, there is a significant difference in the time-scales at which requests are received and processed at a packet processing system and at a server cluster. A cluster-based web server generally receives and processes at most hundreds or a few thousand requests per second. A packet processing system

supporting multiple gigabit ports, on the other hand, receives and processes hundreds-of-thousands to millions of packets per second. Hence, the adaptation time-scales in packet processing systems are likely to be significantly smaller than conventional server clusters. Unfortunately, because of the high rate of packet arrivals, profiling and predicting workload at fine time-scales is prohibitively expensive. A packet processing system may monitor the traffic at coarser time-scales; however, doing so adversely affects the system's ability to react rapidly to traffic fluctuations (and thereby decreases the realizable adaptation benefits). Thus, monitoring and predicting packet arrivals, and hence determining when processor allocations should be adapted, is significantly more challenging in packet processing systems.

Second, most network processors today support multiple processor cores, each with multiple hardware threads (or contexts). For instance, Intel®'s IXP2800 network processor support 16 processors cores with 8 hardware threads each (with a total of 128 hardware threads). If hardware threads are the units of allocation, then such network processor architectures can support processor allocations with $g < 0.01$. As our results indicate, at these g values, a packet processing system can achieve much of the adaptation benefits. Realizing these benefits, however, is challenging. This is because each processor core in these network processors is configured with only a small amount of instruction store (e.g., 4K instructions in IXP2800). The limited instruction store sizes make it impossible to provision hardware threads on each processor with the code for processing all packet types. Thus, a run-time adaptation system is required to determine the mapping of code for subsets of packet types onto different threads/processors such that the instruction store constraints are not violated, and yet the computational requirements for all packet types are met. This mapping is a function of the workload; with fluctuations in the workload the mapping may need to be adapted. Further, to minimize the impact of run-time adaptations on system performance, the derivation of the mapping from packet types to processors and the transitioning of the system from one allocation state to another must be performed within a small fraction of adaptation time-scale. The design of such a light-weight run-time adaptation system poses significant challenges.

5 Conclusion

Most packet processing applications receive and process multiple types of packets. Today, the processors available within packet processing systems are allocated to these packet types at design time. In this paper, we explore the benefits and challenges of adapting processor allocations at run-time in packet processing systems. We demonstrate that, for all the packet processing applications and traces considered, run-time adaptation reduces energy consumption by 70-80% and processor provisioning level by 40-50%. The adaptation benefits are maximized if processor allocations can be adapted at fine time-scales and if processing power can be allocated in small granularities; of these two factors, allocating processors in small granularity is more important—if processing power can be allocated only at coarse granularities, even a very fine adaptation time-scale yields meager benefits. We argue that realizing these adaptation benefits requires the design of a low-overhead, adaptive run-time environment for packet processing systems.

References

[1] Intel IXA Software Developers Kit 3.0. http://www.intel.com/design/network/products/npfamily/sdk3.htm.

[2] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures. In *Proceedings of International Symposium on Microarchitecture*, December 2000.

[3] R. Berrendorf and B. Mohr. PCL - The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors. http://www.fz-juelich.de/zam/PCL/doc/pcl/pcl.pdf.

[4] A. Chandra, P. Goyal, and P. Shenoy. Quantifying the Benefits of Resource Multiplexing in On-demand Data Centers. In *Proceedings of the First Workshop on Algorithms and Architectures for Self-Managing Systems*, June 2003.

[5] J. S. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[6] S. Choi and J. Turner. Configuring Sessions in Programmable Networks with Capacity Constraints. In *Proceedings of IEEE ICC*, May 2003.

[7] K. Egevang and P. Francis. The IP Network Address Translator (NAT). IETF RFC 1631, May 1994.

[8] M. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[9] J. Flinn and M. Satyanarayanan. Energy-aware Adaptation for Mobile Applications. In *Proceedings of Symposium on Operating Systems Principles*, December 1999.

[10] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol Version 3.0. Internet Draft, November 1996.

[11] M. Huang, J. Renau, and J. Torrellas. Positional Adaptation of Processors: Application to Energy Reduction. In *Proceedings of International Symposium on Computer Architecture*, June 2003.

[12] N. C. Hutchinson and L. L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1), 1991.

[13] Internet Protocol. IETF RFC 791, September 1981.

[14] *Intel IXP2400 Network Processor Hardware Reference Manual*, January 2003.

[15] V. Jacobson. Compressing TCP/IP Headers for Low-speed Serial Links. IETF RFC 1144, February 1990.

[16] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3), August 2000.

[17] R. Kokku, T. L. Riché, A. Kunze, J. Mudigonda, J. Jason, and H. M. Vin. A Case for Run-time Adaptation in Packet Processing Systems. Technical Report TR-03-27, Department of Computer Sciences, The University of Texas at Austin, November 2003.

[18] NLANR Network Traffic Packet Header Traces. http://pma.nlanr.net/Traces/.

[19] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of 18th ACM Symposium on Operating Systems Principles*, October 2001.

[20] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott. What TCP/IP Protocol Headers Can Tell Us About the Web. In *Proceedings of ACM SIGMETRICS 2001/Performance 2001*, June 2001.

[21] G. Tsirtsis and P. Srisuresh. Network Address Translation - Protocol Translation (NAT-PT). IETF RFC 2766, February 2000.