Adaptive Load Sharing for Network Processors

Lukas Kencl, Jean-Yves Le Boudec

Abstract

A novel scheme for processing packets in a router is presented that provides load sharing among multiple network processors distributed within the router. It is complemented by a feedback control mechanism designed to prevent processor overload. Incoming traffic is scheduled to multiple processors based on a deterministic mapping. The mapping formula is derived from the robust hash routing (also known as the highest random weight - HRW) scheme, introduced in K.W. Ross, IEEE Network, 11(6), 1997, and D.G. Thaler et al., IEEE Trans. Networking, 6(1), 1998. *No state information* on individual flow mapping has to be stored, but for each packet, a mapping function is computed over an *identifier vector*, a predefined set of fields in the packet. An *adaptive extension* to the HRW scheme is provided to cope with biased traffic patterns. We prove that our adaptation possesses the *minimal disruption property* with respect to the mapping and exploit that property to minimize the probability of flow reordering. Simulation results indicate that the scheme achieves significant improvements in processor utilization. A higher number of router interfaces can thus be supported with the same amount of processing power.

Index Terms

Router architecture, packet processing, load sharing, load balancing, feedback control.

I. Introduction

A. Router Architecture

WITH recent developments in transmission technologies, more demanding performance characteristics are being sought when designing routers. The previously centralized router devices with a single general-purpose processor cannot cope with the ever-increasing workloads and are being replaced by routers with more effective architectures, i.e. distributed or parallel [CAL99], [TW01], [Cha02].

In the case of a *distributed architecture* [Cef97], most of the packet processing load is shifted to special-purpose processors, often called network processors or forwarding engines, typically located directly at the router inputs. Such an architecture has the drawback of poor utilization because all the processors are hardly ever saturated, as the load is almost never evenly distributed over the inputs and does not always reach the nominal rate. *Parallel router architectures* [ADJK92], [Fed00] are based on a pool of parallel processors, located remotely from the inputs, with all of the processors being able to perform the data path processing tasks. Packets may be buffered at the inputs, and relevant fields of the packet (for example, the packet header) are being sent to the pool for resolution. Such an architecture does not suffer from underutilization because loads of all the inputs are combined at the pool. Instead, the

pool interconnect tends to become a major bottleneck. Another drawback is that if load balancing is performed over the pool, the load balancing device is a single point of failure for the entire router.

Other successful designs [ea98], [Sem99] seek to combine both approaches by containing remotely located (at a different switch port than the input line cards) network processors or forwarding engines, which serve a certain predefined set of inputs to carry out the packet processing tasks on packets arriving at these inputs. Again, the traffic may not be evenly distributed over these sets, which leads to less efficient utilization.

We present a novel packet processing scheme, which seeks to *maximize the number of router interfaces* that can be supported with a *fixed amount of network processors of given processing power* while keeping the advantages and avoiding the drawbacks of the aforementioned router architectures. Our basic premise is that a router that provides *load sharing among the network processors* is able to support a greater number of interfaces, while upholding the performance guarantees.

The packet processing tasks are carried out by multiple distributed processors, and packets are scheduled among them according to a mapping computed at run-time. Thus, the total load of the router system is shared among the multiple processing units. The subsequent increase in processor utilization lowers the total system cost and the electricity power consumption. In addition, router fault tolerance is improved.

B. Load Sharing

For a general survey of load-sharing algorithms, see [SHe95]. A widely accepted taxonomy of load-sharing algorithms has been presented by Casavant and Kuhl [CK88]. Eager, Lazowska and Zahorjan [ELZ86] have studied specific adaptive load-sharing policies consisting of a transfer and a location policy. Their work shows that simple adaptive load-sharing policies yield significant performance improvements relative to the non-load-sharing case and, at the same time, performance very close to complex adaptive policies. In addition, a threshold-based location policy is shown to bring substantial improvements over a random selection location policy.

The task of determining a processing unit on which a specific processing job should be executed such that a system-wide function is optimized has been shown to be $\mathcal{NP}$-complete in general (see [ERAL95]). A heuristic that produces the answer in less time, but is not necessarily an optimal one, is thus typically used. Such a global task-scheduling heuristic usually takes some kind of dynamic processor workload information as input. The most effective representation of the workload index has been a topic of intensive research. Kunz [Kun91] has demonstrated that a single, one-dimensional workload descriptor yields better results than more complex descriptors.

In the networking domain, particular attention to load sharing has recently been drawn to the areas of Web servers, Web caching and clustered digital libraries [BO00], [GH99], [Ros97], [ZYZ+98]. The CARP distributed caching scheme, which uses the highest random weight (HRW) algorithm [Ros97] by Ross, is a popular choice for Web caches and is implemented in products offered by Microsoft [BO00]. Although the algorithm provides load balancing over the request object space, it is *not adaptive* and therefore potentially vulnerable to traffic locality.

IBM Network Dispatcher [GH99] is a software tool that routes TCP connections to multiple servers that share their workload, based on a monitored load metric. The algorithm contains an adaptive control loop, but it is required to *maintain state information* where each TCP connection has been mapped.

Load-sharing methods have also recently been studied in relation to the task of distributing Internet traffic over multiple links or paths within the network [CWZ00], [SRS99]. In [CWZ00], the performance of various static hashing schemes as well as of one adaptive scheme for splitting traffic among multiple links is evaluated. The adaptive method, as presented, requires considerable state information to be maintained and can potentially disrupt the flow order. The work presented in [SRS99] concentrates on mapping traffic onto multiple network paths in order to achieve better bandwidth utilization and routing stability. The method divides traffic flows into short-lived and long-lived flows and uses a different mapping discipline for each group: an adaptive one for the long-lived and a static one for the short-lived flows. It is demonstrated that for this problem, thanks to the particular length distribution of network flows, such a hybrid approach is better than each method alone, as it achieves a better balance and saves on signalling overhead. The study of flow length distribution in [SRS99] has been inspirational for some of the experiments presented in Section V.

Other research ([KTZ92], [TZ92]) has concentrated on exploring the possibilities of parallel implementations of the TCP/IP packet processing within routers. In these studies, functional decomposition of individual packet-processing tasks has been determined and various possible forms of parallelism have been categorized: spatial parallelism, pipelining or concurrent operation.

According to this classification, the specific kind of parallelism employed in the load-sharing algorithm presented here would best be characterized as spatial parallelism, i.e., packets are scheduled to multiple processors, all of which are capable of carrying out the same tasks (although they do not necessarily possess homogeneous processing capacity). A mapping is established between flows and processors. It is based on the CARP HRW [Ros97] mapping, extended by an *adaptive control loop*. As in the Network Dispatcher concept [GH99], flows are mapped to processors, yet no state information on particular flows is stored. The HRW mapping is hash-based and is thus easily computable at high speeds (as opposed to, for example, a table-based lookup or classification). The mapping possesses several advantages over other hash-based load balancing schemes; it allows the hashed objects to be split into hash buckets of arbitrary size, as determined by predefined weights. As we prove in this work, a specific method for the weights' adaptation can be found, which results in minimal disruption of the mapping. *Optimization* and *adaptation* of the mapping is the subject of this work.

The mapping adaptation procedure aims to prevent individual processor overload. The design is complicated by the need to minimize the probability of packet reordering within one flow. Owing to the nature of networking transport protocols, it is often illegal—or at least extremely undesirable—to allow packet reordering within a packet flow [KLS98]. Although the widely used TCP protocol attempts to tackle this problem by correct reordering at the destination, reordering slows data delivery, increases receiver buffer size and still may not prevent undesirable retransmissions

and subsequent network congestion.

If packets from the same flow are to be processed by different processors, packet reordering can easily occur. Therefore, packets belonging to a particular flow should be processed by the same processor. As it is not possible to monitor all traffic characteristics in a router, including per-flow state, nor to solve the $\mathcal{NP}$-complete mapping problem at run-time, a fully optimal mapping is not achievable. However, we show that the heuristic presented here, which uses aggregate traffic monitoring as feedback, closely approaches an optimal solution.

C. Outline

The paper is organized as follows: in Section II, we describe the environment and the related assumptions. In Section III, we present the scheme for load sharing among network processors, and in Section IV we lay the theoretical basis for the dynamic adaptation of the scheme by proving the minimal disruption property of our adjustments and then describe the adaptation in detail. In Section V, we present results of our simulations and discuss optimality issues. Section VI deals with aspects of practical implementation of the load-sharing scheme within a router. Finally, in Section VII, we present some concluding remarks.

II. Notation and Assumptions

We consider a router model where certain processors are dedicated to the data plane and certain ones to the control plane. We use the term *Network Processor Unit (NPU)* to denote the device that performs the packet-processing tasks (such as address lookup, classification, filtering, etc.), i.e. the processor dedicated to the data path within a router. In contrast, we denote as *Control Point (CP)* a typically general purpose processor that performs the router control functions such as shortest path computation, topology information dissemination or traffic engineering. Our work concentrates on issues primarily related to the data path within a router.

The router consists of n input-output line cards, m NPUs and at least one CP. With respect to NPUs we consider a heterogeneous router model, where each processor may have different processing power. Thus, μ_j denotes the processing power of NPU j, that is, the maximum number of packet-processing units an NPU j is able to carry out per time unit Δt. We denote μ the total system processing power, that is, $\mu = \sum_1^m \mu_j$.

We denote $\lambda_j(t)$ as the actual packet-processing load of NPU j, that is, the amount of packet processing carried out at NPU j during the interval $(t - \Delta t, t\,]$. We denote $\lambda(t)$ as the total processing load of the system within the time interval, that is, $\lambda(t) = \sum_1^m \lambda_j(t)$. We define $\rho_j(t)$ as the workload intensity of each NPU, that is, $\rho_j(t) = \lambda_j(t)/\mu_j$, and $\rho(t)$ the total system workload intensity, $\rho(t) = \lambda(t)/\mu$.

We denote $\gamma_i(t)$ as the amount of packets that arrive at line card i in time interval $(t - \Delta t, t\,]$. The maximum transport capacity of each link is $\hat{\gamma}$, thus, $\forall i,\ \hat{\gamma} \geq \gamma_i(t)$.

We define the *packet information vector* $\vec{w} = (w_1, w_2, \ldots, w_{k_w})$ as the set of k_w packet fields that are examined, processed or altered within a router and that carry the information based on which the subsequent next-hop of the packet and the treatment applied to the packet within a router are determined (i.e., for example, the destination address,

the source port, TTL, URL, label, etc.). We denote W as the packet information vector space, i.e. the vector space consisting of all possible values of packet information vector $\vec{w} \in W$.

A packet containing an information vector $\vec{w}$ consumes $l(\vec{w})$ processing units at an NPU. We define as *arrival vector* $\vec{a}(t) = (a_{\vec{w}_1}(t), \ldots, a_{\vec{w}_{|W|}}(t))$ a vector of size $|W|$, where the element $a_{\vec{w}}(t)$ denotes the number of packets containing the information vector $\vec{w}$ that arrived at a router during a time interval $(t - \Delta t, t\,]$. Thus $\sum_1^n \gamma_i(t) = \sum_{\vec{w}} a_{\vec{w}}(t)$ and $\lambda(t) = \sum_{\vec{w}} a_{\vec{w}}(t)\, l(\vec{w})$.

We denote as flow *identifier vector* $\vec{v} = (v_1, v_2, \ldots, v_{k_v})$ a set of predefined packet fields that do not change within a particular flow. Each v_i represents a piece of data within the packet and the integer k_v, $k_v \geq 1$, represents the number of fields contained in vector $\vec{v}$. Typically, but not necessarily, $\vec{v}$ is composed of some fields contained within the packet header. For our purposes, any predefined set of fields (or just one of them) that remains constant within a flow can serve as the identifier vector. In this work we assume that $\vec{v} \subseteq \vec{w}$. We denote V as the vector space corresponding to all the possible values of the identifier vector $\vec{v}$ (once the format of the identifier vector has been established).

A typical example of an identifier vector is the traditional flow ID, which consists of a 5-tuple of protocol number (prot), source and destination ports (SP, DP) and source and destination addresses (SA, DA), that is, in such a case, $k_v = 5$ and $\vec{v} =$ (prot, SP, SA, DP, DA). Alternatively, one could use the destination address as a unique parameter, thus $\vec{v} = (v_1) =$ (DA). In the first case, V would represent a set of all possible flow IDs, whereas in the second case, V would be equal to the protocol address space.

Let us define as *identifier persistence vector* $\vec{\Delta}(t) = (\Delta_{\vec{v}_1}(t), \ldots, \Delta_{\vec{v}_{|V|}}(t))$, $\Delta_{\vec{v}}(t) \in \{0, 1\}$ a vector that monitors the persistence of a certain flow (determined by an identifier vector) within a time interval $(t - 2\Delta t, t\,]$. We consider a flow persistent if in each of the two consecutive time intervals $(t - 2\Delta t, t - \Delta t\,]$ and $(t - \Delta t, t\,]$ a packet belonging to the flow arrives. We assume that only persistent flows are vulnerable to reordering, which can occur when consecutive packets belonging to a persistent flow are processed by different processors.

We define time interval ΔT to be the maximum time a single packet spends in the system. If no packet of a flow arrives during the time interval $(t - \Delta T, t\,]$, we assume that processing a subsequent packet from the flow at any processor does not lead to reordering.

In our scenario, we assume that any processor $j \in \{1, \ldots, m\}$ is able to process any packet.

III. Load Sharing for Network Processors

A. Requirements

With the above router model in place, our objectives presented in Section I-A can be reformulated as follows: given a router containing a set of m network processors of processing powers μ_j and given a maximum line card speed $\hat{\gamma}$, maximize the number of interfaces n that such a router can support with a performance constraint $P\,(\text{packet loss}) < \epsilon_p$, where ϵ_p is a given constant.

Definition 1 provides a useful reference point for achieving the objective.

Def. 1: Let us define as **acceptable load sharing** a scheme distributing the interfaces' load among the network processors with the following properties:

- if $\lambda(t) \leq \mu$, then $\forall j,\ \lambda_j(t) \leq \mu_j$, i.e., if the system *is not* overloaded, then *none* of the individual processors is overloaded,
- if $\lambda(t) > \mu$, then $\forall j,\ \lambda_j(t) > \mu_j$, i.e., if the system *is* overloaded, then *all* of the individual processors are overloaded.

Generally, P (packet loss) $= \sum_{j=1}^{m} P\ (\lambda_j(t) > \mu_j)$. In the case of acceptable load sharing, a single processor is overloaded if and only if the entire system is overloaded, thus P' (packet loss) $= P\ (\lambda(t) > \mu) = P\ (\sum_{j=1}^{m} \lambda_j(t) > \sum_{j=1}^{m} \mu)$. Clearly, P' (packet loss) $\leq P$ (packet loss) and P'(packet loss) is the *minimal achievable packet loss probability*. Thus, as acceptable load sharing minimizes the packet loss probability for a given total system load, it enables the total system load to be increased to the uppermost limit possible within the above packet loss probability constraint. Thus the number of supportable interfaces is maximized.

In addition to performance guarantees, a load-sharing system among parallel NPUs should possess the following properties:

Flow order preservation—packet reordering could occur if packets belonging to the same flow were processed by different NPUs. Thus, the assignment of packets to processors should either be fully deterministic with respect to flows, or should attempt to minimize the probability of packets belonging to the same flow being treated by different processors.

Absence of state information—keeping state information upon assigning of concurrent flows is extremely costly in terms of memory and processing overhead. Therefore, it is highly desirable that the assignment of flows to processors can be carried out without the state information being stored.

Support of heterogeneous processors—the system must be able to support heterogeneous architectures, that is, where there are processors with various processing capacities present or where preference should be given to some processors as to the amount of requests processed.

Fault tolerance—the system must be able to adjust to a processor failure quickly and gracefully, i.e. without a great disruption.

B. Packet-to-NPU Mapping

The basis of our load-sharing scheme is that the load of each input (ingress traffic arriving at a line card) is distributed for processing among the NPUs using a *deterministic* mapping f (see Figure 1). The mapping f is computed over the *identifier vector* $\vec{v}$. The computation $f(\vec{v}) = j$ determines the particular NPU j to which the packet is mapped for processing. The function $f(\vec{v})$, $f : V \rightarrow \{1, 2, \ldots, m\}$, splits the vector space V into m exclusive subspaces V_j. Packets from a particular subspace are all mapped to the same processor.

Upon arrival of a packet at an input, the packet is parsed to extract the fields relevant for packet processing, i.e., the identifier vector $\vec{v}$ and the packet information vector $\vec{w}$. The packet is buffered, the mapping $f(\vec{v})$ is computed and the

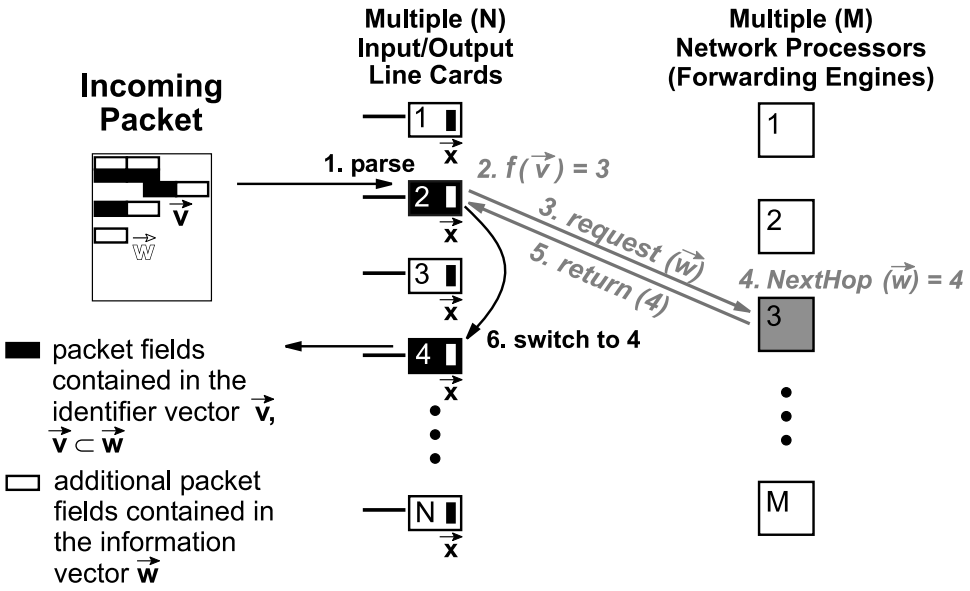


Fig. 1. **Load-sharing scheme abstraction.** Upon arrival, a packet is parsed to extract the relevant fields, the identifier vector $\vec{v}$ and the packet information vector $\vec{w}$. Then, the packet is buffered, the mapping $f(\vec{v})$ is computed and the packet information vector $\vec{w}$ is sent for resolution to NPU j, $f(\vec{v}) = j$. At NPU j, the vector $\vec{w}$ is processed and the resolution information is returned to the requesting unit. The packet is then switched to the correct output port and the corresponding packet alterations or manipulations, based on the resolution results, are applied.

packet information vector $\vec{w}$ is then sent to NPU j, $f(\vec{v}) = j$ for resolution.

At NPU j, the packet information vector $\vec{w}$ is processed and the resolution information about the treatment to be applied to the packet (next hop, outgoing switch port, QoS applied) is returned to the requesting unit. Then, the packet is switched to the correct outgoing port and the corresponding packet alterations or manipulations, based on the resolution results, are applied (this may mean, for example, applying certain QoS, attaching an MPLS label or splicing with another TCP connection).

The mapping f we propose for such a purpose is based on the robust hash mapping scheme (alternatively called highest random weight (HRW) mapping) presented in [TR98] and extended in [Ros97].

Def. 2: **Packet-to-NPU (HRW) Mapping f:** Let $g(\vec{v}, j)$ be a pseudo-random function $g : V \times \{1, 2, \ldots, m\} \rightarrow (0, 1)$, i.e., we assume $g(\vec{v}, j)$ to be a random variable in $(0, 1)$ with uniform distribution. Let a packet arrive at an input i, carrying an identifier vector $\vec{v} \in V$. The mapping $f(\vec{v})$ is then computed as follows:

$$f(\vec{v}) = j \tag{1}$$

$$\Leftrightarrow$$

$$x_j \, g(\vec{v}, j) = \max_{k \in \{1,\ldots,m\}} x_k \, g(\vec{v}, k), \tag{2}$$

where $x_j \in \mathbb{R}^+$ is a weight multiplier assigned to each NPU.

The weights $\vec{x} = (x_1, \ldots, x_m)$, as described in [Ros97], have a 1-to-1 correspondence with the partitioning vector $\vec{p} = (p_1, \ldots, p_m)$, which determines the fraction of request object space (the identifier vector space V, in our case) assigned for processing to each NPU, i.e., $p_j = |V_j|/|V|$.

The HRW mapping possesses the following properties, which are particularly useful for the purpose of flow-to-processor mapping [Ros97], [TR98]:

Load balancing—the robust hash mapping provides load balancing over the request object space, even for the heterogeneous case. This is extremely useful for the ability to support processors of heterogeneous processing capacities because the mapping weights allow the fraction of load mapped to a particular processor to be controlled.

Minimal disruption—it has been shown in [TR98] that in the case of a processor failure, removal or addition, the number of request objects that are remapped to another destination is minimal. This property is useful for providing *fault tolerance* (if a particular processor fails, only flows mapped to that processor are affected).

However, we observe that the minimal disruption property is not limited to these special cases. In Section IV we show that by a similar line of proof as in [TR98], the minimal disruption property holds as well for *certain special kinds of adjustments* of the mapping weights. We exploit that fact when carrying out the mapping adaptation in order to *minimize the amount of flow remappings* caused by the adaptation.

For load-sharing purposes in general, there is no need for the mapping f to be identical at all line cards. In fact, a different mapping can be used at each line card. For example, at line card i, a mapping $f_i(\vec{v})$ could be computed using function g_i of the form $g_i(\vec{v}, j) = g(\vec{v}, i + j)$. However, our scheme does require that the weights vector $\vec{x}$ be identical at each card.

IV. Adaptation through Feedback

A. Problem Statement

Load sharing among multiple processors can become very inefficient if insufficient attention is paid to keeping the individual processor load under control. The goal of the adaptation is to prevent undesirable effects, i.e. primarily processor overload and consequent packet loss. It may not be obvious how such effects can occur when, as claimed, the HRW mapping provides load balancing. However, it is important to note that it provides load balancing over the *request object space*, i.e., in our case, the identifier vector space V. In contrast, the loads due to the actual traffic received at the router input ports may by no means be distributed uniformly over this request object space, but rather will exhibit certain locality patterns. This means that in spite of the load-balancing property, mapping f can potentially lead to grossly imbalanced load distributions. For such cases, the mapping must be adjusted to account for non-uniform load distribution in the received traffic. As thus the mapping f now changes with time, we define $f_{(t)}(\vec{v}) : V \rightarrow \{1, 2, \ldots, m\}$ as the instance of f at time t.

The objective of the control loop is to prevent overutilization of a single processor when the system is under-utilized or, vice versa, to prevent under-utilization of a single processor when the system is over-utilized. At the same time, we aim to minimize the amount of packet-to-NPU remappings. Assuming that the mapping f is being adjusted periodically in time intervals Δt, the objective for the adjustment at time $t - \Delta t$ can be formulated as the following optimization problem:

Def. 3: **NPU load-sharing optimization problem:**

$$\max \sum_{\vec{v} \in V} \Delta_{\vec{v}}(t) \sum_{j \in M} 1\{f_{(t-\Delta t)}(\vec{v}) = j\} * 1\{f_{(t)}(\vec{v}) = j\}, \tag{3}$$

with constraints:

$$\text{if } \rho(t) \leq 1 \quad \Rightarrow \quad \lambda_j(t) \leq \mu_j, \forall j, \tag{4}$$

$$\text{if } \rho(t) > 1 \quad \Rightarrow \quad \lambda_j(t) \geq \mu_j, \forall j, \tag{5}$$

where

$$\lambda_j(t) = \sum_{\vec{v} \in V} 1\{f_{(t)}(\vec{v}) = j\} \sum_{\vec{w} \supseteq \vec{v}} a_{\vec{w}}(t)\, l(\vec{w}). \tag{6}$$

Note that this problem statement is only useful for defining the objective of our method, but not for computing the actual solution $f_{(t)}$. In order to be able to carry out the optimization described in Def. 3, one would have to know already at time $t - \Delta t$ both $\Delta_{\vec{v}}(t)$, which would require one to maintain a huge amount of state information, as well as $a_{\vec{w}}(t)$, which can only be predicted speculatively. Furthermore, even if such knowledge were available, one would still have to solve an $\mathcal{NP}$-complete problem, as Def. 3 is an integer linear programming optimization. Thus, heuristics, such as the one presented below, are typically used, yet the above definition remains useful for setting the objective and evaluating the quality of the solution ex-post.

Note that in order to apply the formula for minimizing the packet-to-NPU remappings to the minimization of the number of packets reordered, the adaptation interval Δt must satisfy $\Delta t \geq \Delta T$. Then, during the interval a flow is vulnerable to reordering, ΔT, a packet flow can be remapped at most once and thus minimizing the number of remappings also minimizes the number of reorderings.

B. Adaptation Algorithm

The adaptation scheme works in the following general way (see Figure 2): periodically, the CP gathers information about the workload intensity of the NPUs. If an adaptation threshold is exceeded, the CP adjusts the weights of the mapping f. The new multiplicative weights vector $\vec{x}$ is then downloaded to the NPUs. Let $\vec{x}(t)$ be the instance of weights' vector $\vec{x}$ used to compute $f(t)$ at time t.

In order to evaluate the status of individual processors, we need a processor workload intensity indicator. For that purpose, we introduce a smoothed, low-pass *filtered processor workload intensity* measure $\bar{\rho}_j(t)$ of the form

$$\bar{\rho}_j(t) = \frac{1}{r}\, \rho_j(t) + \frac{r-1}{r}\, \bar{\rho}_j(t - \Delta t), \tag{7}$$

where r is an integer constant. A similar filtered measure for total system workload intensity is introduced as $\bar{\rho}(t) = \frac{1}{r}\, \rho(t) + \frac{r-1}{r}\, \bar{\rho}(t - \Delta t)$. The filtering is done to reduce the influence of short-term load fluctuations and to obtain information about the *trend* in processor workload intensity.

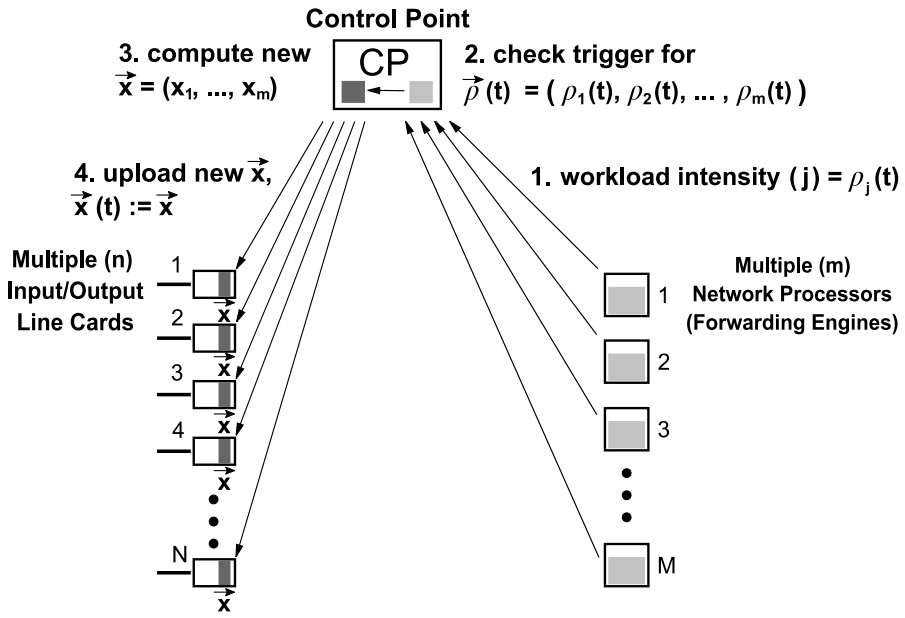


Fig. 2. **Load sharing with feedback.** Periodically, the CP gathers information about the workload intensity of the NPUs $\vec{\rho}(t) = (\rho_1(t), \rho_2(t), ..., \rho_m(t))$. If the adaptation is triggered, the CP adjusts the multiplicative weights vector $\vec{x}$ and the new vector is then downloaded to the NPUs.

The adaptation algorithm consists of two parts (see Figure 3): the *triggering policy*, which specifies the conditions to act, and the *adaptation policy*, which specifies how to act. A trigger is periodically evaluated and, based on the result, specific action is taken.

1) Triggering Policy: We introduce a dynamic *workload intensity threshold* $\epsilon'_\rho(t)$ defined as

$$\epsilon'_\rho(t) = \bar{\rho}\,(t) + \frac{1}{2}\,(1 - \bar{\rho}\,(t)) \tag{8}$$

$$= \frac{1}{2}\,(1 + \bar{\rho}\,(t)). \tag{9}$$

Thus the dynamic workload intensity threshold is positioned midway between the current filtered total system workload intensity $\bar{\rho}(t)$ and workload intensity of 1. The closer the total system workload intensity approaches 1, the higher the likelihood of violating the acceptable load-sharing bounds and therefore the tighter the threshold follows the total system workload intensity.

During time intervals when the total system workload intensity $\bar{\rho}(t)$ remains in the vicinity of 1, the value of the workload intensity threshold may be too close to $\bar{\rho}(t)$ to provide a meaningful threshold for adaptation. To prevent such cases, we introduce a form of *hysteresis* into the threshold computation by defining a fixed threshold in the close vicinity of 1.

Let $\epsilon_h > 0$ be a fixed hysteresis bound, which prevents adaptation within the interval $((1 - \epsilon_h)\,\bar{\rho}(t),\ (1 + \epsilon_h)\,\bar{\rho}(t))$. The ϵ_h is typically set to a value close to 0, for example 0.01, thus preventing adaptation when the load stays within 1 percent of the total system workload intensity.

A dynamic triggering threshold $\epsilon_\rho(t)$, which combines the workload intensity threshold $\epsilon'_\rho(t)$ with the hysteresis

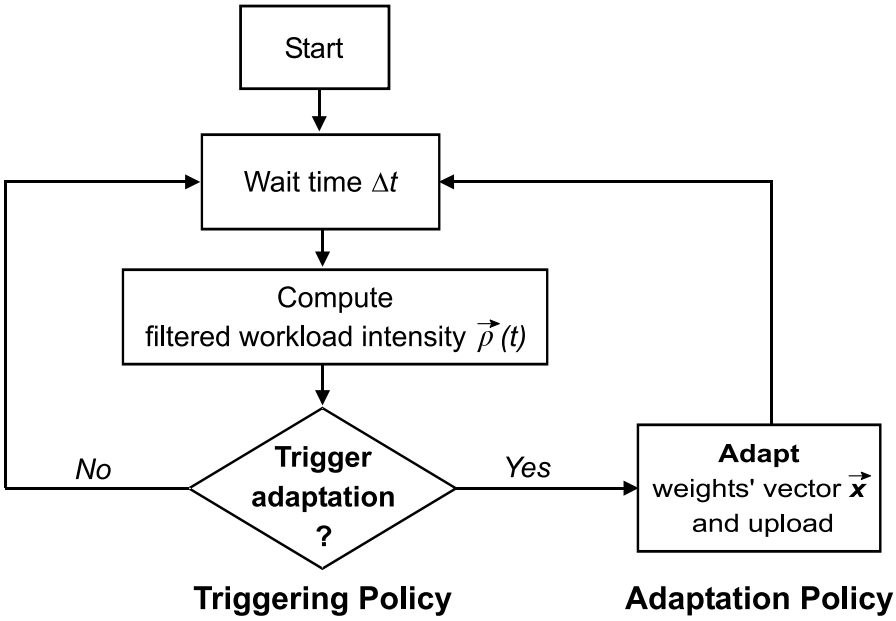


Fig. 3. Adaptation algorithm.

bound, is thus set to determine the amount of over- (or under-)utilization allowed at one processor:

Def. 4: **Triggering Threshold $\epsilon_\rho(t)$:** Let the dynamic workload intensity threshold $\epsilon'_\rho(t)$ and the hysteresis bound ϵ_h be defined as above. Then the triggering threshold is defined (according to whether the system in total is over- or underutilized) as follows:

$$\epsilon_\rho(t) = \max\ (\epsilon'_\rho(t),\ (1+\epsilon_h)\,\bar{\rho}(t)), \quad \bar{\rho}(t) \leq 1, \tag{10}$$

$$\epsilon_\rho(t) = \min\ (\epsilon'_\rho(t),\ (1-\epsilon_h)\,\bar{\rho}(t)), \quad \bar{\rho}(t) > 1. \tag{11}$$

The result of the comparison of the filtered workload intensity to the threshold then acts as a trigger for the adaptation to start. An appropriate trigger is again chosen according to whether the system as a whole is over- or underutilized:

$$\begin{aligned} \bar{\rho}(t) \leq 1 \quad &\Rightarrow \quad \text{if } (\epsilon_\rho(t) < \max_j \bar{\rho}_j(t)) \text{ then } \textit{adapt} \\ \bar{\rho}(t) > 1 \quad &\Rightarrow \quad \text{if } (\epsilon_\rho(t) > \min_j \bar{\rho}_j(t)) \text{ then } \textit{adapt}. \end{aligned}$$

2) Adaptation policy: We propose a simple scheme for the periodic adaptation that operates directly on the weights' vector $\vec{x}$. Propositions 1 and 2 provide the theoretical basis:

Proposition 1: Let $\alpha \in \mathbb{R}^+$, $\alpha \neq 1$. Let A, B be two nonempty, mutually exclusive subsets of $M = \{1, \ldots, m\}$, $M = A \cup B$. Let f, f' be two HRW mappings using identical pseudo-random function $g(\vec{v}, j)$, but having different weight vectors $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{x}\,' = (x'_1, \ldots, x'_m)$ as follows:

$$x'_j = \alpha\, x_j, \quad j \in A, \tag{12}$$

$$x'_j = x_j, \quad j \in B. \tag{13}$$

Let p_j and p'_j, denote the fraction of request object space mapped to node j using the HRW mapping with weights $\vec{x}$

and $\vec{x}\,'$, respectively. Then, if $\alpha < 1$,

$$p'_j \leq p_j, \;\; j \in A \tag{14}$$

$$p'_j \geq p_j, \;\; j \in B. \tag{15}$$

and, conversely, if $\alpha > 1$,

$$p'_j \geq p_j, \;\; j \in A \tag{16}$$

$$p'_j \leq p_j, \;\; j \in B. \tag{17}$$

Proof: We first prove the inequality (14) by contradiction. Assume that $\exists j_0 \in A$ such that $p'_{j_0} > p_{j_0}$. It means that there exists at least one identifier vector $\vec{v}_0$, for which $f'(\vec{v}_0) = j_0$, yet $f(\vec{v}_0) \neq j_0$.

As $f'(\vec{v}_0) = j_0$, we have $x'_j\, g(\vec{v}_0, j_0) = \max_{k \in M}\; x'_k\, g(\vec{v}_0, k)$. But then:

$$\begin{aligned}
x_{j_0} g(\vec{v}_0, j_0) &= \frac{1}{\alpha}\, x'_{j_0} g(\vec{v}_0, j_0) \\
&\geq \frac{1}{\alpha}\, x'_k g(\vec{v}_0, k) \\
&= x_k g(\vec{v}_0, k), \quad \forall k \in A; \\
x_{j_0} g(\vec{v}_0, j_0) &= \frac{1}{\alpha}\, x'_{j_0} g(\vec{v}_0, j_0) \\
&\geq \frac{1}{\alpha}\, x'_k g(\vec{v}_0, k) \\
&= \frac{1}{\alpha}\, x_k g(\vec{v}_0, k) \\
&\geq x_k g(\vec{v}_0, k), \quad \forall k \in B.
\end{aligned}$$

Therefore, $x_{j_0} g(\vec{v}_0, j_0) = \max_{k \in M} x_k g(\vec{v}_0, k)$ and thus $f(\vec{v}_0) = j_0$, which contradicts our assumption.

Inequality (15) can be proved in a symmetrical way, as well as the case of $\alpha > 1$. □

Equality in inequalities (14)-(17) is an extreme case, which can only take place if α is so close to 1 that the weights $\vec{x}$ change so little that the change does not affect any single identifier vector.

Note that, given the complex relationship between vectors $\vec{x}$ and $\vec{p}$ (see [Ros97]), it is hard to make further general statements about the effects of making direct adjustments of $\vec{x}$.

Proposition 2—Minimal disruption: Let $\alpha \in \mathbb{R}^+$. Let A, B be two nonempty, mutually exclusive subsets of $M = \{1, \ldots, m\}$, $M = A \cup B$. Let f, f' be two HRW mappings using the identical pseudo-random function $g(\vec{v}, j)$, but differing in the weight vectors $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{x}\,' = (x'_1, \ldots, x'_m)$ as follows:

$$x'_j = \alpha\, x_j, \quad j \in A, \tag{18}$$

$$x'_j = x_j, \quad j \in B. \tag{19}$$

Let p_j and p'_j denote the fraction of request object space mapped to node j using the HRW mapping with weights $\vec{x}$ and $\vec{x}\,'$, respectively. Then, the fraction of request object space mapped to two different nodes by the two mappings

is equal to $\frac{1}{2}\sum_j |p_j - p'_j|$. In other words, the amount of request objects mapped by the two mappings to different destinations is minimal.

Proof: The case of $\alpha = 1$ is trivial. Let $\alpha < 1$. We prove that for each node j, exactly $|p_j - p'_j|\,|V|$ objects have changed the mapping. The proof is divided into two parts:

1) $j \in A$: from Proposition 1 we know that $p'_j \leq p_j$. Let us show that all objects mapped to j by f' are also mapped to j by f by contradiction: assume that there exists at least one identifier vector $\vec{v}_0$, for which $f'(\vec{v}_0) = j$, yet $f(\vec{v}_0) \neq j$. But, if $f'(\vec{v}_0) = j$, this means that $x'_j\, g(\vec{v}_0, j) = \max_k\ x'_k\, g(\vec{v}_0, k)$ and therefore

$$\begin{aligned} x_j g(\vec{v}_0, j) &= \frac{1}{\alpha}\, x'_j g(\vec{v}_0, j) && (20)\\ &\geq \frac{1}{\alpha}\, x'_k g(\vec{v}_0, k) && (21)\\ &\geq x_k g(\vec{v}_0, k), \quad \forall k \in M. && (22) \end{aligned}$$

   Thus, $x_j\, g(\vec{v}_0, j) = \max_k\ x_k\, g(\vec{v}_0, k)$ and $f(\vec{v}_0) = j$, which contradicts our assumption. As all objects mapped to j by f' are also mapped to j by f, the amount of request objects in which the two mappings differ at node j is equal to the fraction $|p_j - p'_j|$ of request object space.

2) $j \in B$: from Proposition 1 we know that $p'_j \geq p_j$. Let us show that all objects mapped to j by f are also mapped to j by f' by contradiction: assume that there exists at least one identifier vector $\vec{v}_0$, for which $f(\vec{v}_0) = j$, yet $f'(\vec{v}_0) \neq j$. But, if $f(\vec{v}_0) = j$, this means that $x_j\, g(\vec{v}_0, j) = \max_k\ x_k\, g(\vec{v}_0, k)$ and therefore

$$\begin{aligned} x'_j g(\vec{v}_0, j) &= \alpha\, x_j g(\vec{v}_0, j) && (23)\\ &\geq \alpha\, x_k g(\vec{v}_0, k) && (24)\\ &\geq x'_k g(\vec{v}_0, k), \quad \forall k \in M. && (25) \end{aligned}$$

   Thus $x'_j\, g(\vec{v}_0, j) = \max_k\ x'_k\, g(\vec{v}_0, k)$ and $f'(\vec{v}_0) = j$, which contradicts our assumption. As all objects mapped to j by f are also mapped to j by f', the amount of request objects in which the two mappings differ at node j is equal to the fraction $|p_j - p'_j|$ of request object space.

Thus, the two mappings differ by $|p_j - p'_j|\,|V|$ vectors at each node, which leads to a fraction of $\frac{1}{2}\sum_j |p_j - p'_j|$ difference in total.

The proof for $\alpha > 1$ is symmetrical. □

It is important to note that the minimal disruption property would not generally hold for the adaptation if the weights' adjustment was not carried out by a *single constant multiplier*, as then the inequalities (21) and (24) would not necessarily hold for all $k \in M$. As the minimal disruption property is crucial for minimizing the amount of remappings, Propositions 1 and 2 serve as a background for designing the adaptation of vector $\vec{x}$ to be carried out by a single, constant multiplier:

Def. 5: **Weights-Vector $\vec{x}$ Adaptation:** Let $\bar{\rho}(t) \leq 1$. Assuming that the trigger condition $(\epsilon_\rho(t) < \max_j \bar{\rho}_j(t))$

is satisfied, let

$$c(t) = \left(\frac{\epsilon_\rho(t)}{\min \{\bar{\rho}_j(t) \mid \bar{\rho}_j(t) > \epsilon_\rho(t)\}} \right)^{1/m}. \tag{26}$$

Then

$$x_j(t) := \quad c(t)\, x_j(t - \Delta t), \quad \bar{\rho}_j(t) > \epsilon_\rho(t), \tag{27}$$

$$x_j(t) := \qquad x_j(t - \Delta t), \qquad \bar{\rho}_j(t) \leq \epsilon_\rho(t). \tag{28}$$

Conversely, the adaptation for the case of $\bar{\rho}(t) > 1$ is performed in a symmetrical manner.

Thus, in the case that the system is underutilized, the presented adaptation *lowers the weights for the exceedingly* (with respect to a threshold) *overutilized processors*, whereas weights for others remain unchanged. Conversely, if the system in total is overutilized, the adaptation *raises the weights for the exceedingly* (with respect to a threshold) *underutilized processors*. The lowering or raising of weights is carried out proportionally, either to the minimal workload intensity $\bar{\rho}_j(t)$, which exceeds the threshold $\epsilon_\rho(t)$, or to the maximal workload intensity $\bar{\rho}_j(t)$ which remains below the threshold $\epsilon_\rho(t)$.

The factor $1/m$ in the exponent of $c(t)$ represents the effects of the number of processors present—less aggressive adjustment is needed if there are more processors. Alternatively, the exponent of the form of $1/\log_2(m)$ can be used, making the root computation less complex. The effects of using either exponent are evaluated in Sections V-E and V-F.

V. Numerical Results

We have used the MATLAB v.6 environment on an IBM PC Pentium III with Microsoft Windows 2000 machine to simulate a model of a router with multiple NPUs and line cards.

A. Simulations Input

For router input, we have used generated traffic. The parameters for generating the per-interface traffic were approximated from OC-3 traces statistics compiled in [Aix00], [Nla97] and [TMW97] and approximated to OC-192 speed by shortening the time intervals proportionally, i.e., 1 second of the monitored OC-3 traffic corresponds to 15 ms in our OC-192-like traces. Note that this transformation is a simplification of reality, since the scaled traces would differ not only along the time dimension, but the per-flow data volume, the multiplexing effects and the packet interarrival times would have to be taken into account as well.

The following parameters characterize the traffic:

- *Number of packets arrived per time interval*—a discrete time homogeneous Markov chain, attaining values in the interval of [3000, 22000] with uniform transition probability to states within a neighboring interval, the step of change limited within [−4000, 4000] packets of difference at each iteration every 15 ms.
- *Number of flows existent in a time interval*—a stochastic recurrence, with state space attaining values in the interval of [8000, 240000], with uniform transition probability to states within a neighboring interval. The step

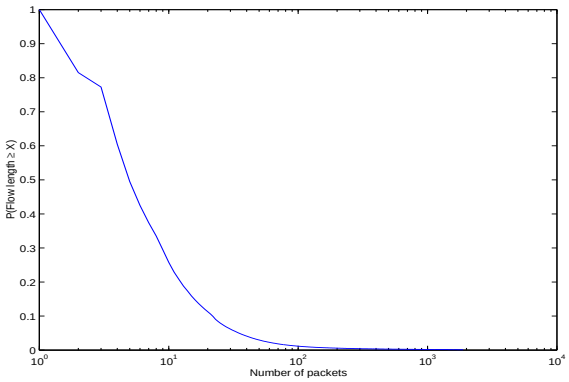


Fig. 4. Flow length cumulative distribution.

of change in the amount of flows at each iteration every 15 ms ranges between a positive and negative value of the maximum amount of packets per iteration divided by the mean number of packets per flows finished in the previous iteration. The direction of change (increase or decrease) in the number of flows is correlated with the direction of change in the number of packets (number of flows grows when number of packets grows and vice versa), as shown in [TMW97] to hold.

- *Flow length*—the amount of packets in a flow. We have used a measured flow length distribution as documented in [SRS99]. The distribution is shown in Fig. 4. As the measured distribution covers only a finite part of the spectrum (up to a flow length equal to 1886 packets), we have used a Pareto distribution to approximate the heavy tail of the distribution. Other publications on the topic of flow length distribution [TMW97] [ZBPS02] [CTB98], as well as the analysis of [Aix00] exhibit similar patterns and thus we believe it to be a good approximation.
- *Maximum per-flow fraction of interface rate* ϵ_f—the maximum fraction of the total rate of an interface (in packets per second) a single flow is allowed to occupy. If $\epsilon_f = 1/2$, a single flow may occupy up to 50% of the interface transport capacity. Note that the study [ZBPS02] suggests that such limits on the maximum per-flow rate are in agreement with reality.
- *Identifier vector values*—for the distribution of identifier vector values, we have approximated a typical distribution of IP source and destination addresses in networking traffic as described in [Nla97]. The prevalence of class C addresses, which occupy a relatively small portion of the address space (12.5%) and yet account for approximately 65% of the packets in network traffic, led us to consider a normal distribution of identifier vectors within a 32-bit integer space, with parameters fitted to those measured in [Nla97]. Thus, the identifier vectors of flows are generated with a truncated normal distribution $\mathcal{N}(0, 1)$ out of the 32-bit integer space.
- *Load per packet*—we have used a simplified representation of the packet load in number of processing units it takes to process a packet. We have defined three possible levels of the per-packet load $l \in \{1, 2, 3\}$. The rationale for such a definition is an analogy with the most common router function—the address lookup—where the forwarding table is often organized into a tree structure and a lookup requires a variable number of memory accesses, depending on the tree depth per particular prefix [DBCP97] [NK98]. Often, the trees are organized in

three levels, and thus an address lookup may require 1-3 memory accesses. In our simulations, the distribution of the per-packet load over the identifier vectors is uniform. That is certainly a simplification of reality, as obtaining realistic values would require matching traffic traces to a corresponding lookup table, a task not within our means.

B. System Model Parameters

The iterations of the traffic generation process as well as the evaluations of the adaptation trigger are carried out at a time interval Δt of 15 ms. Such intervals should be comfortably large to be greater than any hypothetical ΔT (maximum time a packet can spend in the system), which is typically defined in terms of nanoseconds. The unit of load at each NPU is equal to 1 memory access. Unless stated otherwise, we assume a homogeneous router model, where all the NPUs have equal processing capacity, corresponding to a full load of a single router interface, which amounts to 22000 packet per 15 ms. This leads to μ_j = 44000 processing units (memory accesses) per 15 ms. The NPUs are considered buffer-less devices that cannot process more traffic within an interval than the limit of maximum number of processing units per time interval. Traffic exceeding this limit per time interval is assumed to be dropped.

Unless stated otherwise, the low-pass filter constant r is set to $r = 3$ and the hysteresis bound to $\epsilon_h = 0.1$. The number of links n and processors m in the simulations have been chosen such that the total system workload intensity remains close to 1, so that it makes sense to investigate the performance with respect to the acceptable load-sharing bounds.

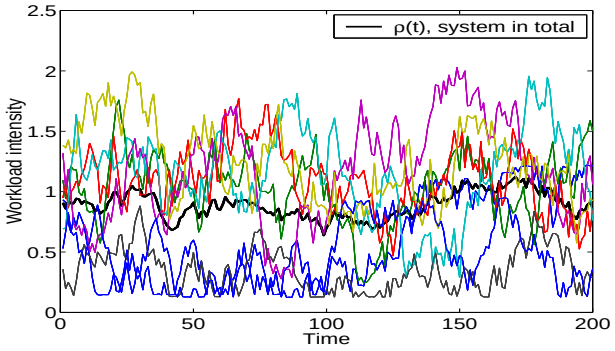
In the subsequent simulations, three alternatives of a router system model are compared frequently to demonstrate the functionality and advantages of the adaptive load-sharing method; (1) a naive system where *no load sharing* is deployed over the processors and thus the entire load of any single interface is mapped to a particular processor; (2) a *static* load-sharing system, which uses the HRW packet-to-NPU mapping f to map packets to processors, yet with mapping weights remaining static, as configured initially according to the capacities of the individual processors; and, (3), the *adaptive* system, with the dynamically adapted weights of the packet-to-NPU mapping $f_{(t)}$.

C. Adaptive Load Sharing

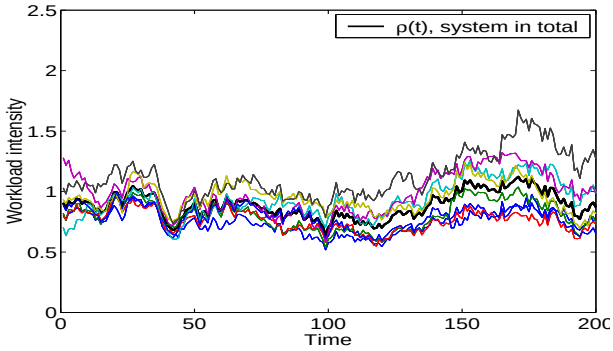
Figures 5–7 illustrate via a simulation how the adaptive load sharing method works in general. A load of $n = 13$ links is processed by a router equipped with $m = 8$ processors. The cases where no load sharing is deployed, where load sharing is deployed using the static mapping and where load sharing is deployed using the adaptive mapping are compared. All the alternatives are put in perspective by comparison with the ideal solution.

Figure 5 compares the per-processor workload intensity for each method. Clearly, individual processor workload intensity remains within close vicinity of the ideal workload intensity when adaptive load sharing is deployed.

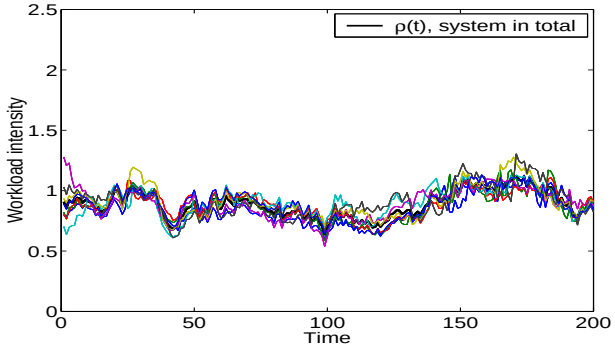
Figure 6 compares the number of packets dropped under the three scenarios. Again, the adaptive case results closely follow the total system curve, which represents the global minimum. In Fig. 7, the number of per-iteration flow remappings for the adaptive load-sharing method is examined. Note that the number of flows remapped per iteration is several orders of magnitude smaller than the number of flows appearing per iteration.



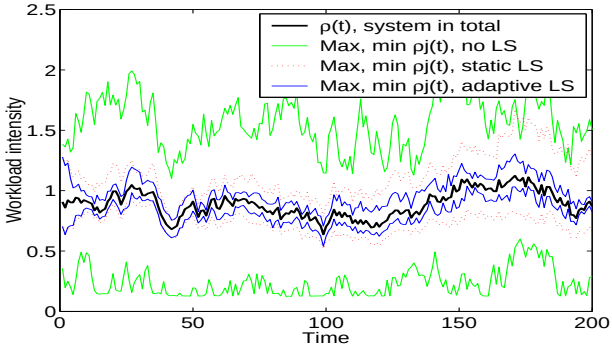
(a) No load sharing.



(b) Static load sharing.



(c) Adaptive load sharing.



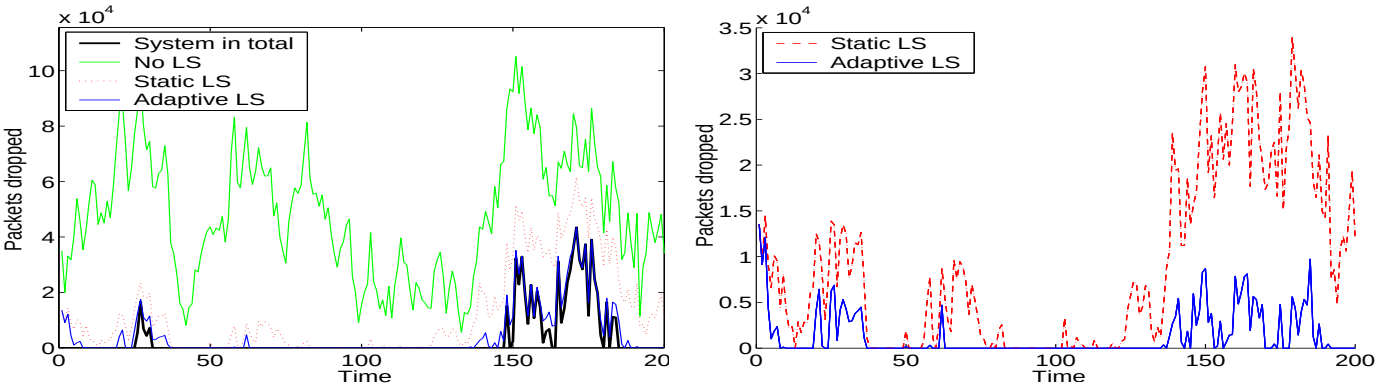
(d) Combined.

Fig. 5. **Per-processor workload intensity.** A load of $n = 13$ links is processed by a router equipped with $m = 8$ processors. If no load sharing is deployed, the entire load of each link is assigned to a particular processor. This is compared to a case where load sharing is deployed using a static, non-adaptive mapping f and to load sharing with the dynamically adapted mapping $f_{(t)}$. Individual processor workload intensity when (a) no load sharing, (b) the static load-sharing, and (c) the adaptive method is deployed. (d) Summary to results: maximum and minimum per-processor workload intensity per each scheme. In all figures, the total system workload intensity $\rho(t)$, which is the same in all three cases, is shown for comparison, as it represents an ideal reference value. Clearly, individual processor workload intensity remains within the closest vicinity of the ideal workload intensity when adaptive load sharing is deployed.

Relevant results of a multitude of such experiments are summarized in Tables I and II.

D. Influence of Maximum Per-Flow Rate ϵ_f

Simulations show that the fraction ϵ_f a single flow is allowed to consume from the rate of a single interface, and, consecutively and more importantly, from the capacity of a single NPU, is a key parameter. It determines the effectiveness of the adaptive method. The lower the limit on a single flow's rate, the better the method performs, as shown in Fig. 8, where the same system model is subject to traffic with a varying maximum flow rate factor: Cases of the maximum flow rate being limited to 10%, 25%, 50% of the link capacity and the case where a single flow can occupy the whole link (or NPU) are compared. Clearly, the lower the limit, the better the adaptive method performs.



(a) Packets dropped in total.

(b) Packets dropped in excess.

Fig. 6. **Packets dropped.** A load of $n = 13$ links is processed by a router equipped with $m = 8$ processors. The graphs depict the numbers of dropped packets when not using a load sharing scheme, when using a static load sharing method and when using an adaptive method. (a) Number of packets dropped by the system using each of the schemes, compared to the ideal value of the minimal number of packets the system would have to drop, should the individual processors be ideally saturated. (b) Comparison of static and adaptive methods in terms of the number of packets dropped in excess of the ideal minimal value. Clearly, the adaptive method significantly outperforms the static one and avoids a large number of unnecessary packet drops, in particular during periods when the ideal solution likewise leads to no packet drops.

E. Fractional Factorial Analysis of the Load-Sharing Method

To test the influence of various tunable parameters (factors) on the adaptive method's performance, we have employed the technique known as fractional factorial analysis [Jai91]. Upon alternating the values of each factor, one can measure the influence of each factor on the variance of the results.

Each factor has been alternated between two values (levels)—high (Hi) and low (Lo). The following factors and levels have been explored:

TABLE I

ACCEPTABLE LOAD SHARING.

Packets dropped	*% of All*
Acceptable load sharing (ideal)	0.7548
Adaptive load sharing	1.3256
Static load sharing	2.2590
Packets dropped in excess of ideal	
Adaptive load sharing	0.5708
Static load sharing	1.5042
Improvement of adaptive over static	%
1 - (adaptive / static)	60.4591

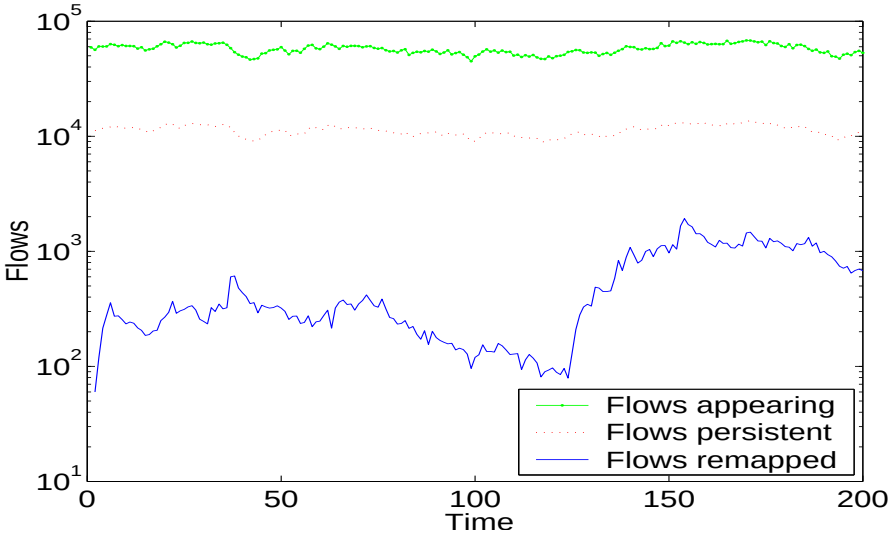


Fig. 7. **Flows remapped.** Load of $n = 13$ links is processed by a router equipped with $m = 8$ processors, with the adaptive load sharing method being executed. Shown on a on logarithmic scale are the statistics of the number of flow remappings that occurs within an iteration. The upper line shows the total number of flows appearing within an iteration, the middle line the number of persistent flows (flows that had appeared in the previous iteration as well) and the lowest one the amount of persistent flows remapped within an iteration. Clearly, there is at least one order of magnitude difference between each of the flow statistics, showing that only a small fraction of flows is vulnerable to remapping and out of these, only a very small fraction is eventually remapped.

- *Adaptation coefficient exponent.* The levels used were $Lo = 1/\log_2(m)$ and $Hi = 1/m$;
- *NPU processing capacity distribution.* For level Lo, all the individual capacities were equal, whereas level Hi implied an exponential distribution of capacities, with relative capacities in $\{1, 2, 4, 8\}$;
- *Adaptation interval.* The levels used were $Lo = 1$, meaning the adaptation would be carried out at every iteration, if triggered, and $Hi = 10$, meaning the adaptation would only be carried out every 10th iteration;
- *Hysteresis bound* ϵ_h. The levels used were $Lo = 0.01$ and $Hi = 0.1$;

TABLE II

FLOW REMAPPINGS.

	All	Persistent	Remapped
Mean over simulations			
% of all	100.00	19.23	0.02
% of persistent	-	100.00	0.11
Per iteration			
Max, # of flows	77,076	15,304	204
Max, % of all	100.00	21.19	0.43
Max, % of pers.	-	100.00	2.22

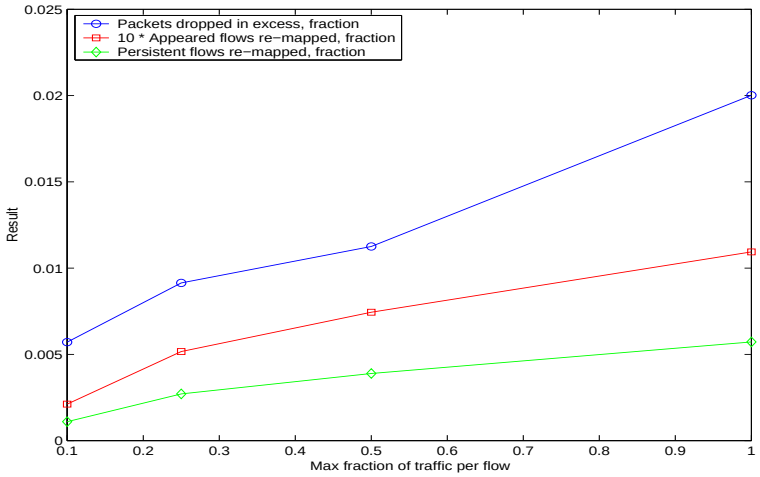


Fig. 8. **Influence of maximum per-flow fraction rate limit ϵ_f.** The key response variables—number of packets dropped in excess, the fraction of flows remapped out of all flows seen per iteration and out of all flows persistent per iteration—are shown for four different settings of the maximum per-flow rate ϵ_f: 10%, 25%, 50% and 100% of the line rate, or of a single processor's capacity. All the observed response variables grow with the increasing ϵ_f, which thus has a negative impact on the method's performance.

- *Filtering parameter* r. The levels used were $Lo = 3$ (meaning that a significant influence is attributed to the recently measured value) and $Hi = 32$ (meaning the recently measured value has less influence).

The results of the experiments are shown in Table III, where -1 represents the Lo level and 1 the Hi level. The results on the impact of each factor on the variance of the key response variables, the number of *packets dropped in excess* and the number of *flows remapped* are shown in Table IV.

The fractional factorial analysis results show that of the factors studied, only three have a significant influence on

TABLE III

FRACTIONAL FACTORIAL ANALYSIS RESULTS.

No.	Exponent	NPU cap.	Adaptation int.	Hysteresis	Filter	*Packets dropped*	*Flows remapped*
1.	-1	-1	-1	1	-1	87091	700
2.	-1	-1	1	-1	1	223181	422
3.	-1	1	-1	1	1	203333	306
4.	-1	1	1	-1	-1	164660	438
5.	1	-1	-1	-1	1	177332	858
6.	1	-1	1	1	-1	228731	190
7.	1	1	-1	-1	-1	119675	926
8.	1	1	1	1	1	266604	16
Total / 8						183830	482

TABLE IV

EFFECTS OF FACTORS.

Factor	*Packets dropped*		*Flows remapped*	
	Estimate	% of Variation	Estimate	% of Variation
Total / 8	183830		482.0	
Adaptation exponent	14260	0.066	15.5	0.003
NPU capacity dist.	4742	0.007	-60.5	0.040
Adaptation interval	36968	0.443	-215.5	0.512
Hysteresis bound ϵ_h	12164	0.052	-179.0	0.354
Filter parameter r	33787	0.370	-81.5	0.073

the results: the adaptation interval, the hysteresis bound and the filtering parameter.

The most significant factor appears to be, not surprisingly, the *adaptation interval*. Clearly, more frequent adaptation leads to more accurate load sharing and thus less packet loss, yet more frequent adaptation leads to more flow remappings as well. The exact duration of the adaptation interval thus needs to be set according to the desired parameters of the system.

The number of packets dropped is likewise significantly influenced by the *filtering parameter* r. As this factor, in comparison, has relatively little influence on the number of flows remapped, it makes sense to choose a value of r that leads to fewer packet drops. Clearly, when r is high, the system does not take the recent information strongly enough into account. A lower value, like the $r = 3$ in the experiment, can thus be recommended.

The second most influential factor on the variance of the number of flow remappings is the *hysteresis bound* ϵ_h. The hysteresis bound, on the contrary, has little effect on the number of packets dropped. It thus clearly makes sense to set the bound to a larger value, for example the $\epsilon_h = 0.1$ used in the experiment, in order to prevent unnecessary flow remappings.

The relatively insignificant influence of the NPU processing capacity distribution factor confirms that the HRW mapping and the weights' adaptation work equally well with both uniform and highly nonuniform distributions of processing capacities. Likewise, the exact value of the adaptation coefficient exponent does not have much influence. This outcome favors using the $\log_2(m)$ option, as computing the root in Eq. (26) is then less demanding.

F. Influence of the Number of Processors

In this simulation we evaluate the effects of altering the total number of processors present in the system, m. We compare systems where the total processing capacity remains constant, yet the number of NPUs present and their processing capacity differs. Here again, the scenarios compared involve a router with $n = 13$ interfaces and $m = 8$ NPUs of uniform processing capacity μ_j, plus a system with $m = 64$ processors, where each NPU has a processing capacity of $\mu_j/8$.

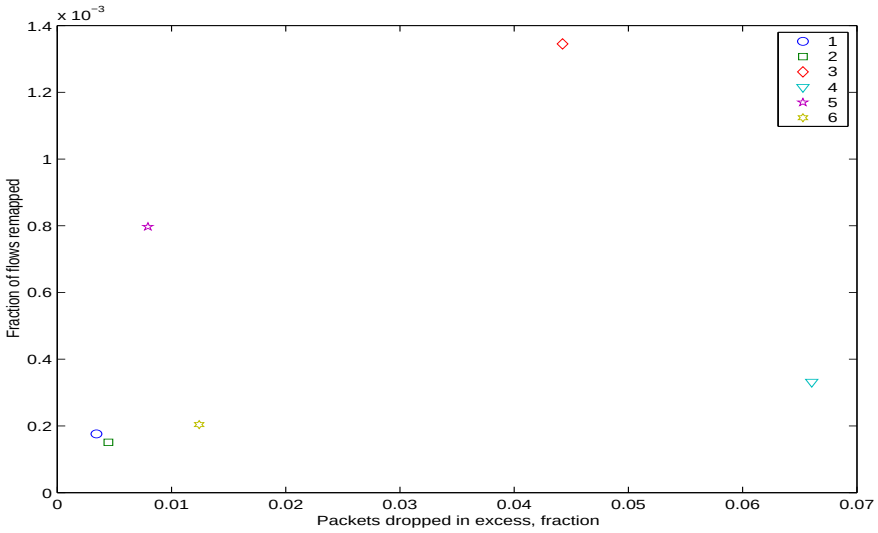


Fig. 9. **Influence of the number of NPUs.** Graphic representation of results in Table V. Experiments 1 and 2 are conducted with $m = 8$ processors and 3-6 with $m = 64$ processors. In 5 and 6, the maximum per-flow rate fraction ϵ_f is adjusted accordingly and thus these two experiments do not differ too significantly from the results of 1 and 2. However, clearly, the higher the number of processors, the more the accuracy of the method is affected.

At the same time we compare the effects of altering the exponent in the adaptation coefficient between the values of $1/m$ and $1/\log_2(m)$ to observe which of the two more accurately reflects the higher number of processors present. Furthermore, we alter the maximum per-flow fraction ϵ_f, because the maximal fraction a single flow is allowed to consume from a single interface's rate is reflected in the maximum fraction of processing capacity a single flow may consume at a single NPU. Thus, in one set of simulations when $m = 64$, we decrease ϵ_f according to the ratio of the decrease of a single NPU's processing capacity, $1/8$. Thus, ϵ_f attains a value of either $\epsilon_f = 0.1$ or $\epsilon_f = 0.0125$.

The individual experiments and the results are summarized in Table V and Fig. 9. Clearly, as shown in Section V-D, the influence of the maximum per-flow rate ϵ_f is crucial for the results of experiments No. 3 and 4. Thus in order to evaluate fairly the effects of the number of processors m, the maximum per-flow rate ϵ_f needs to be adjusted as in

TABLE V

INFLUENCE OF THE NUMBER OF PROCESSORS.

No.	m	Adaptation exponent	Maximum flow rate (%)	Packets dropped in excess (%)	Flows remapped (%)
1	8	$1/\log_2(m)$	10.00	0.343	0.0176
2	8	$1/m$	10.00	0.447	0.0151
3	64	$1/\log_2(m)$	10.00	4.423	0.1345
4	64	$1/m$	10.00	6.605	0.0332
5	64	$1/\log_2(m)$	1.25	0.795	0.0797
6	64	$1/m$	1.25	1.242	0.0204

experiments No. 5 and 6.

When ϵ_f is adjusted, results of the system with $m = 64$ NPUs (No. 5 and 6) differ not as significantly from results on a system with $m = 8$ NPUs, although the performance does worsen in both of the key response variables. The effects of the altered exponents in computing the adaptation coefficients can be compared. Using $1/m$ leads to less aggressive adaptation of the mapping weights, and thus fewer flows are remapped. In contrast, the less fine-grained $1/\log_2(m)$ adjusts the mapping more aggressively and thus prevents packet loss, but at a cost of more flow remappings.

VI. Implementation Issues

A. Router Architecture

An implementation of a router combining the distributed router architecture with the load-sharing scenario is depicted in Figure 10.

Another potential implementation is shown in Figure 11. It is a load-sharing extension of the concepts of the Juniper Networks routers [Sem99], where the header and the payload-processing paths are separated by two switches, input and output. The control information in the packet header is processed in a remote NPU, whereas the payload is temporarily stored in a distributed shared memory coupled to the input switch. Sharing the load among the NPUs brings significant utilization benefits.

B. Packet-to-NPU Mapping

The major implementation issue related to load sharing is how to provide a fast-computable pseudo-random function g for computing the mapping f, with the properties required in the mapping definition (Def. 2).

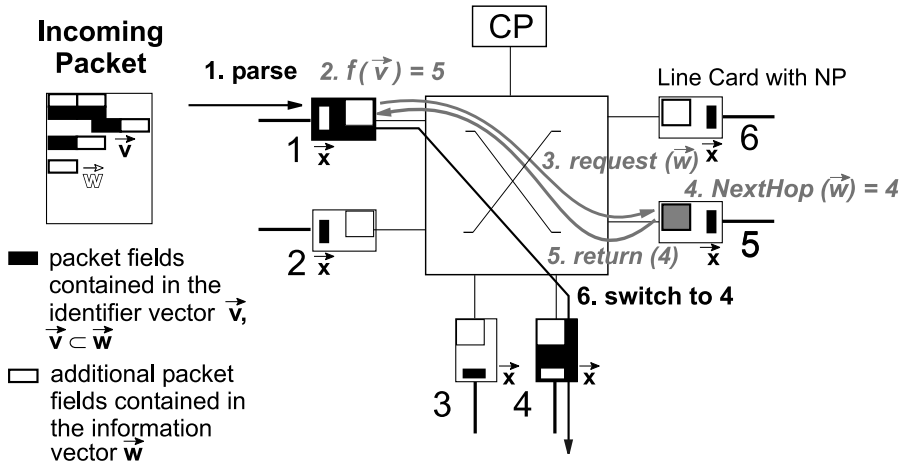


Fig. 10. **Load sharing within a distributed multiprotocol router.** An NPU resides at every line card and the NPUs are interconnected through a switch. Upon packet arrival, the packet information vector $\vec{w}$ travels through the switch to NPU j, chosen by computing the mapping $f(\vec{v}) = j$ over the identifier vector $\vec{v}$. At NPU j, the vector $\vec{w}$ is processed and the resolution information is returned to the requesting NPU. The packet is then switched to the correct output port and the corresponding packet alterations or manipulations, based on the resolution results, are applied. The mapping weights can be adjusted to give preference to local processors.

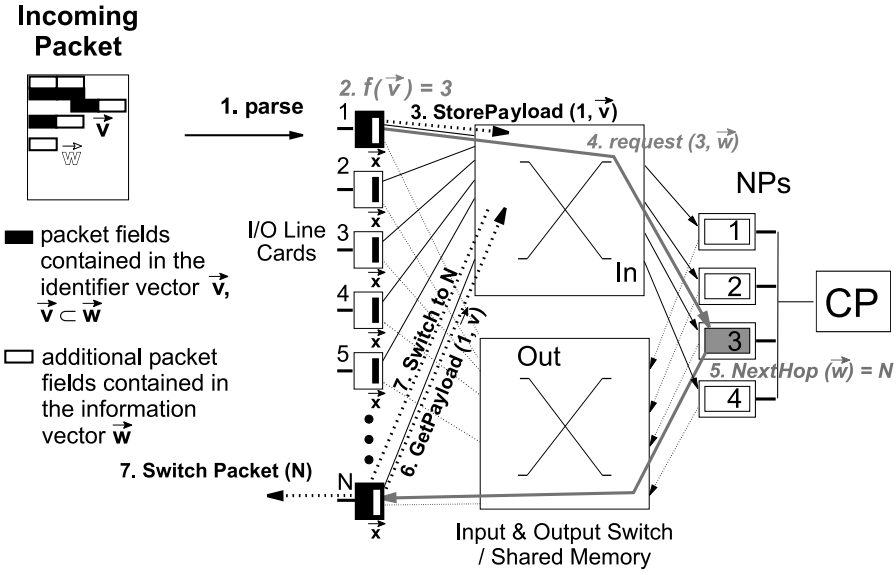


Fig. 11. **Multiprotocol router, consisting of an input- and output switch/shared memory and a pool of multiple NPUs, sharing the load of N line cards.** Every line card is connected to two switches/shared memories, input and output. Each of the multiple NPUs is reachable from a line card through a switch. Upon packet arrival, the packet is buffered in the input shared memory and the information vector $\vec{w}$ travels through the input switch to NPU j, chosen by computing the mapping $f(\vec{v}) = j$ over the identifier vector $\vec{v}$ at the line card. At NPU j, the vector $\vec{w}$ is processed and the resolution information is switched to the resulting port of the output switch. The packet is then retrieved from the shared memory, the corresponding packet alterations or manipulations, based on the resolution results, are applied and the packet is transmitted on the link.

A good candidate appears to be the hash function based on the Fibonacci golden ratio multiplier $\phi^{-1} = (\sqrt{5}-1)/2$ presented in [Knu98]. The Fibonacci hash function leads to the "most random" scrambling of sequences [Knu98]. It is defined as follows:

$$h_{\phi^{-1}}(x) = (\phi^{-1}\, x) \text{mod } 1. \tag{29}$$

Such a function can be fit into the mapping scheme as follows:

$$g(\vec{v}, j) = h_{\phi^{-1}}(\vec{v} \text{ XOR } h_{\phi^{-1}}(j)). \tag{30}$$

As the values $h_{\phi^{-1}}(j)$ can be precomputed, the actual computation per vector $\vec{v}$ requires only $4m$ basic mathematical operations and m comparisons (to find the maximum).

Depending on computing capacity available, other hash functions may be used. The study in [CWZ00] documented good spreading properties of the CRC16 function, but at the cost of higher complexity. For a survey of other hash functions for implementing the HRW mapping, see [RKMD02].

In our experiments, we have used Fibonacci hashing to compute $g(\vec{v}, j)$.

C. Load Indicator

Another open implementation issue is how to actually measure the load of the processors or the number of processing units spent per time interval. Alternative measures to the one used in the simulations (number of memory accesses an

NPU has performed during the time interval Δt) can be for example the number of processing cycles or the number of packets processed per time interval. A counter value is periodically read by the CP. Multiple indicators can be combined into one using relative weights, as in [GH99], yet note that results in [Kun91] indicate that the benefit over a one-dimensional load descriptor is minimal.

Although the load information gathering appears to present unnecessary extra effort for the router system, note that similar statistics collection is readily available and often required from the existing router equipment [(ed95] [MR91] and therefore should result in a minimum load increase on the system.

VII. Conclusions and Future Work

We have proposed a scheme for sharing packet-processing tasks among multiple network processors within a router. The scheme is based on an adaptive deterministic mapping of flows to processors. The proposed load-sharing scheme requires no flow state information to be stored within a router. The mapping itself is derived from the robust hash routing presented in [Ros97] and [TR98]. We have extended the mapping with an adaptation discipline aimed at keeping the processor load below a dynamically derived threshold. The threshold reflects the total system workload intensity.

The adaptation is performed by adjusting the weights of the packet-to-processor mapping, thus reducing or increasing the amount of flows a processor must handle. Thanks to the proved minimum disruption property of our adjustments of the mapping, the adaptation requires only a very small number of flows to be remapped. Thus the probability of packet reordering within a flow is kept low.

Such a scheme is particularly useful in routers with many input ports, and with packets requiring large amounts of processing. With the proposed scheme, a kind of statistical multiplexing of the incoming traffic over the multiple network processors is achieved, thus in effect transforming a router into a parallel computer. The improvements in processor utilization decrease the total router cost and power consumption as well as improve fault tolerance.

The spectrum of potential applications is not limited to a router. The method can be deployed in any networking system that benefits from spreading the load over multiple processing units and that requires packets belonging to a single flow to be processed by the same processor. Such applications include, for example, a Web server load balancer or a distributor of traffic over multiple network links.

As for future improvements, further study is planned to gain insight into how various traffic patterns influence the performance of the load-sharing scheme. Another topic open for research is the influence of load sharing on QoS guarantees provided by a router.

VIII. Acknowledgement

We thank Anees Shaikh for providing the statistical data on flow length distribution.

REFERENCES

[ADJK92] A. Asthana, C. Delph, H. V. Jagadish, and P. Krzyzanowski. Towards a Gigabit IP router. *Journal of High Speed Networks*, 1(4):281–288, 1992.

[Aix00] AIX–MAE West interconnection at NASA Ames OC-3 trace. National Laboratory for Applied Network Research (NLANR), March 19 2000. http://moat.nlanr.net/PMA/.

[BO00] G. Barish and K. Obraczka. World Wide Web caching: trends and techniques. *IEEE Communications Magazine*, 38(5):178–184, May 2000.

[CAL99] H. C. B. Chan, H. M. Alnuweiri, and V. C. M. Leung. A framework for optimizing the cost and performance of next-generation IP routers. *IEEE Journal on Selected Areas in Communications*, 17(6):1013–1029, June 1999.

[Cef97] Cisco Express Forwarding (CEF). Cisco Systems white paper, 1997. http://www.cisco.com.

[Cha02] H. J. Chao. Next generation routers. *Proceedings of the IEEE*, 90(9):1518–1558, September 2002.

[CK88] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988.

[CTB98] M. Crovella, M. Taqqu, and A. Bestavros. *A Practical Guide To Heavy Tails*, chapter 1: Heavy-Tailed Probability Distributions in the World Wide Web, pages 3–26. Chapman & Hall, 1998.

[CWZ00] Z. Cao, Z. Wang, and E. W. Zegura. Performance of hashing-based schemes for Internet load balancing. In *Proceedings of the INFOCOM'00 Conference*, pages 332–341, 2000.

[DBCP97] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small forwarding tables for fast routing lookups. *ACM Computer Communication Review*, 27(4):3–14, October 1997.

[ea98] C. Partridge et al. A fifty gigabit per second IP router. *IEEE/ACM Transactions on Networking*, 6(3):237–248, June 1998.

[(ed95] F. Baker (editor). Requirements for IP version 4 routers. RFC 1812, Internet Engineering Task Force, June 1995.

[ELZ86] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogenous distributed systems. *IEEE Transactions on Software Engineering*, 12(5):662–675, May 1986.

[ERAL95] H. El-Rewini, H. H. Ali, and T. Lewis. Task scheduling in multiprocessing systems. *IEEE Computer*, 28(12):27–37, December 1995.

[Fed00] G. C. Fedorkow. Cisco 10000 Edge Services Router (ESR) technology overview, 2000. http://www.cisco.com.

[GH99] G. Goldszmidt and G. Hunt. Scaling internet services by dynamic allocation of connections. In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pages 171–184, May 1999.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[KLS98] V. P. Kumar, T. V. Lakshman, and D. Stilliadis. Beyond best effort: router architectures for the differentiated services of tommorrow's Internet. *IEEE Communications Magazine*, pages 152–164, May 1998.

[Knu98] D. E. Knuth. *Sorting and searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1998.

[KTZ92] O. G. Koufopavlou, A. N. Tantawy, and M. Zitterbart. Analysis of TCP/IP for high performance parallel implementations. In *17th IEEE Conference on Local Computer Networks*, Minneapolis, September 1992.

[Kun91] T. Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, 17(7):725–730, July 1991.

[MR91] K. McCloghrie and M.T. Rose. Management information base for network management of TCP/IP-based internets:MIB-II. RFC 1213, Internet Engineering Task Force, March 1991.

[NK98] S. Nilsson and G. Karlsson. Fast address lookup for Internet router. In *Proceedings IFIP 4th International Conference on Broadband Communications '98*, pages 11–22, 1998.

[Nla97] WAN traffic distribution by address size, Fix-West trace. National Laboratory for Applied Network Research (NLANR), May 1997.
http://www.nlanr.net/NA/Learn/Class.

[RKMD02] R. Russo, L. Kencl, B. Metzler, and P. Droz. Scalable and adaptive load balancing on IBM PowerNP. Research Report RZ 3431, IBM Zurich Research Laboratory, July 2002.

[Ros97] K. W. Ross. Hash routing for collections of shared web caches. *IEEE Network*, 11(6):37–44, November-December 1997.

[Sem99] Ch. Semeria. Internet backbone routers and evolving Internet design. Juniper Networks white paper, September 1999.
http://www.juniper.net.

[SHe95] B. A. Shirazi, A. R. Hurson, and K. M. Kavi (editors). *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE CS Press, 1995.

[SRS99] A. Shaikh, J. Rexford, and K. G. Shin. Load-sensitive routing of long-lived IP flows. In *Proceedings of SIGCOMM '99*, September 1999.

[TMW97] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–27, November-December 1997.

[TR98] D. G. Thaler and C. V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking*, 6(1):1–14, February 1998.

[TW01] J. Turner and T. Wolf. Design issues for high performance active routers. *IEEE JSAC*, 19(3):404–409, March 2001.

[TZ92] A. Tantawy and M. Zitterbart. Multiprocessing in high performance IP routers. In *Proceedings of the 3rd IFIP WG 6.1/6.4 Workshop on Protocols for High Speed Networks*, Stockholm, May 1992.

[ZBPS02] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. In *Proceedings of ACM Sigcomm '02*, Pittsburgh, August 2002.

[ZYZ+98] H. Zhu, T. Yang, Q. Zheng, D. Watson, O. H. Ibarra, and T. Smith. Adaptive load sharing for clustered digital library servers. In *Proceedings of the Seventh International Symposium on High Performance Distributed Computing*, pages 235–242, July 1998.

PLACE PHOTO HERE

Lukas Kencl (lukas.kencl@ieee.org) received a Master's degree in Computer Science from the Charles University, Prague, Czech Republic, in 1995. He continued his education in 1997-98 at the Graduate School of Communication Systems at the Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland. Upon successful completion, he joined the Department of Communication Systems at the IBM Zurich Research Laboratory. In parallel, he continued his studies at EPFL, where he obtained a Ph.D. degree in Communication Networks in 2003. His research interests include load-sharing algorithms, router architecture, network processors and computer networking in general.

PLACE PHOTO HERE

Jean-Yves Le Boudec (jean-yves.leboudec@epfl.ch) is full professor at EPFL and director of the Institute of Communication Systems. He graduated from Ecole Normale Superieure de Saint-Cloud, Paris, where he obtained the Agregation in Mathematics (rank 4) in 1980 and received his doctorate in 1984 from the University of Rennes, France. From 1984 to 1987 he was with INSA/IRISA, Rennes. In 1987 he joined Bell Northern Research, Ottawa, Canada, as a member of scientific staff in the Network and Product Traffic Design Department. In 1988, he joined the IBM Zurich Research Laboratory where he was manager of the Customer Premises Network Department. In 1994 he joined EPFL as associate professor. His interests are in the architecture and performance of communication systems.