Adaptive Load Sharing for Network Processors

Lukas Kencl, Jean-Yves Le Boudec

Abstract—**A novel scheme for processing packets in a router is presented, which provides for 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 needs 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 in order 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 in order 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.**

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 could not cope with the ever-increasing workloads and are being replaced by routers of more effective architectures, distributed or parallel [6].

In the case of a *distributed architecture* [5], 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* [2], [9] 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 under-utilization 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 [16], [18] 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 a 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 which provides for *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 [19]. A widely accepted taxonomy of load sharing algorithms has been presented by Casavant and Kuhl [4]. Eager, Lazowska and Zahorjan [7] 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 no 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 so that a system-wide function is optimized has been shown to be NP-complete in general (see [8]). A heuristic, which produces the answer in less time, but 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 [14] has demonstrated that a single, one-dimensional workload descriptor yields better results than more complex descriptors.

In the networking domain, particular interest in load sharing has recently been raised in the areas of Web servers, Web caching and clustered digital libraries [3], [10], [17], [23]. The CARP distributed caching scheme, which uses the highest random weight (HRW) algorithm [17] by Ross, is a popular choice for Web servers and is implemented in products offered by Microsoft [3]. 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 [10] 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 infor-*

L. Kencl is with the IBM Zurich Research Laboratory, Rüschlikon, Switzerland. E-mail: lke@zurich.ibm.com .

J.-Y. Le Boudec is full professor at the Department of Communication Systems (DSC), Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. E-mail: jean-yves.leboudec@epfl.ch .

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

mation where each TCP connection has been mapped.

Other research ([12], [20]) 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, which are all 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 [17] mapping, extended by an *adaptive control loop*. As in the Network Dispatcher concept [10], 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 to split the hashed objects into hash buckets of arbitrary size, as determined by predefined weights, and, as we prove in this work, a specific method for the weights' adaptation can be found, which results in only a minimum 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. Due to the nature of networking transport protocols, it is often illegal, or at least extremely undesirable, to allow packet reordering within a packet flow [13]. Although the widely used TCP protocol attempts to tackle this problem by correct reordering at the destination, reordering slows down data delivery, increases receiver buffer size and still may not prevent some 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 the full traffic characteristics in a router, including per-flow state, nor to solve the NP-complete mapping problem at run-time due to performance limits of the current devices, a fully optimal mapping is not achievable. However, we show that the heuristic presented here, which uses aggregate traffic monitoring as feedback, stays within small bounds from the 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 the 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 different processors are dedicated to the data plane and to the control plane. We use the term *Network Processor (NPU)* to denote the device performing the packet processing tasks (such as address lookup, classification, filtering, etc.), that means, the processor dedicated to the data path within a router. In contrast, we denote as *Control Point (CP)* a 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, by μ_j we denote 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, $\mu = \sum_1^m \mu_j$.

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

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

As the *packet information vector* $\vec{w} = (w_1, w_2, \ldots, w_{k_w})$ we define 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 as W 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 remain constant within a flow can serve as the identifier vector. In this work we assume that $\vec{v} \subseteq \vec{w}$. By V we denote the vector space corresponding to all the possible values of the identifier vector $\vec{v}$ (once the format of the identifier vector is established).

A typical example of an identifier vector would be the traditional flow ID, consisting 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 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, if packets of these flows are processed by different processors. 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 (packet drop) $< \epsilon_p$, 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 drop) $= \sum_{j=1}^{m} P\ (\lambda_j(t) > \mu_j)$. In the case of acceptable load sharing, a single processor is overloaded if and only if the entire system is overloaded, thus P' (packet drop) $= P\ (\lambda(t) > \mu) = P\ (\sum_{j=1}^{m} \lambda_j(t) > \sum_{j=1}^{m} \mu)$. Clearly, P' (packet drop) $\leq P$ (packet drop) and P'(packet drop) is the *minimal achievable packet drop probability*.

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

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 [21] and extended in [17].

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

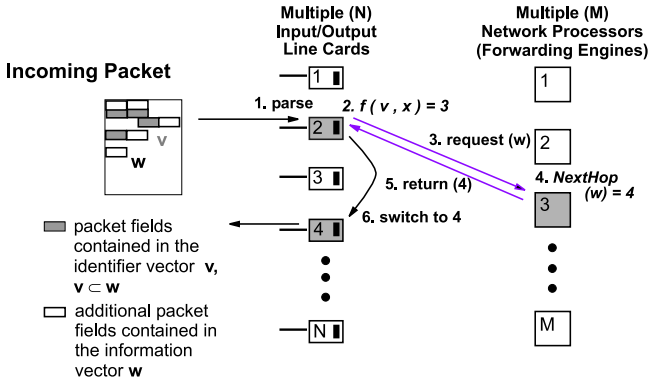

Fig. 1. Load sharing scheme abstraction.

as follows:

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

$$\Leftrightarrow$$

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

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 [17], are in 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 [17], [21]:

Load balancing–the robust hash mapping provides load balancing over the request object space, even for the heterogeneous case. That 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 [21] that in the case of a processor failure, removal or addition, the number of request objects that are re-mapped 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 [21], 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 re-mappings* caused by the adaptation.

For the 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 attention is not paid to keeping the individual processor load under control. The goal of the adaptation is to prevent undesirable effects, mainly, processor overload and a consequent packet drop. 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. That 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 the non-uniform load distribution in the received traffic.

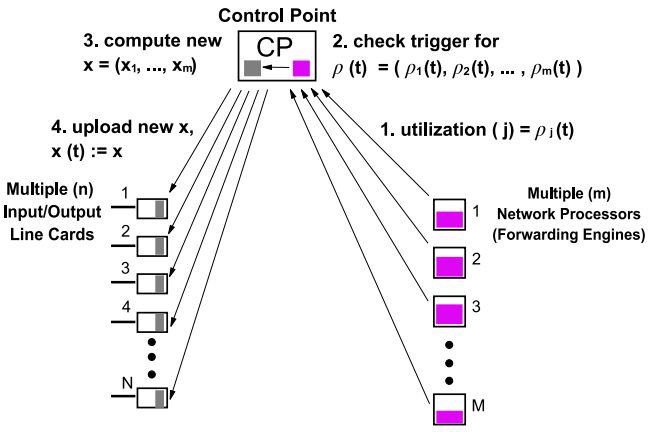

Fig. 2. Load sharing with feedback.

The objective of the control loop is to prevent over-utilization 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 re-mappings. Thus, the objective can be formulated as the following optimization problem:

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

$$\max \sum_{\vec{v} \in V} \Delta_{\vec{v}}(t) \sum_{j \in M} 1_{\{f(t-\Delta t)(\vec{v})=j\}}\, 1_{\{f(t)(\vec{v})=j\}}, \quad (3)$$

with constraints:

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

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

where

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

B. Adaptation Algorithm

The adaptation scheme works in the following general way (see Figure 2): periodically, the CP gathers information about the utilization 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.

As the mapping f now changes with time, we define $f(t) : V \to \{1, 2, \ldots, m\}$ as the instance of f at time t and $\vec{x}(t)$ as the instance of weights' vector $\vec{x}$ used to compute $f(t)$.

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

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

where r is an integer constant. A similar filtered measure for total system utilization 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

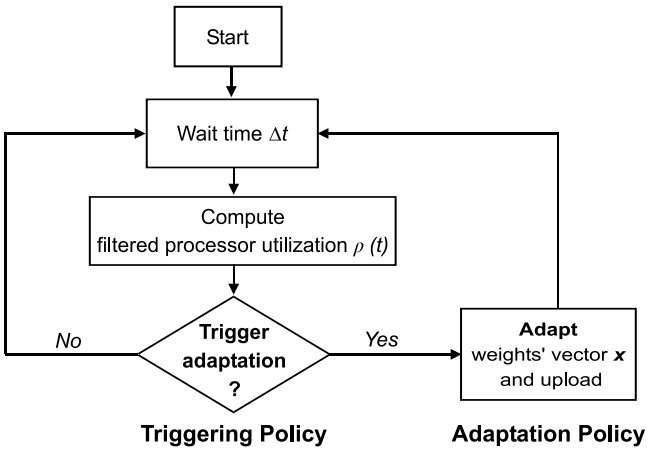

Fig. 3. Adaptation algorithm.

load fluctuations and to obtain information about the *trend* in processor utilization.

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.

B.1 Triggering Policy

We introduce a dynamic *utilization threshold* $\epsilon'_\rho(t)$ defined as

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

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

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

During time intervals when the total system utilization $\bar{\rho}(t)$ remains in the vicinity of 1, the value of the utilization 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 from being carried out within the interval $\big((1 - \epsilon_h)\,\bar{\rho}(t),\ (1 + \epsilon_h)\,\bar{\rho}(t)\big)$. 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 utilization.

A dynamic triggering threshold $\epsilon_\rho(t)$, which combines the utilization threshold $\epsilon'_\rho(t)$ with the hysteresis bound, is thus set for determining the amount of over- (or under-) utilization allowed at one processor:

Def. 4: **Triggering Threshold $\epsilon_\rho(t)$:** Let the dynamic utilization 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 under-utilized) as follows:

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

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

The result of the comparison of the filtered utilization to the threshold then acts as a trigger for the adaptation to start. An appropriate trigger is again chosen according to whether the system in total is over- or under-utilized:

$$\bar{\rho}(t) \le 1 \quad \Rightarrow \quad \text{if } (\epsilon_\rho(t) < \max_j \bar{\rho}_j(t)) \text{ then } \textit{adapt}$$

$$\bar{\rho}(t) > 1 \quad \Rightarrow \quad \text{if } (\epsilon_\rho(t) > \min_j \bar{\rho}_j(t)) \text{ then } \textit{adapt}.$$

B.2 Adaptation policy

We propose a simple scheme for the periodic adaptation, operating 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 differing in the weight vectors $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{x}\,' = (x'_1, \ldots, x'_m)$ as follows:

$$x'_j = \alpha\, x_j, \quad j \in A, \quad (12)$$

$$x'_j = x_j, \quad j \in B. \quad (13)$$

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 $\alpha < 1$,

$$p'_j \le p_j, \ \ j \in A \quad (14)$$

$$p'_j \ge p_j, \ \ j \in B. \quad (15)$$

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

$$p'_j \ge p_j, \ \ j \in A \quad (16)$$

$$p'_j \le p_j, \ \ j \in B. \quad (17)$$

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:

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

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 is a contradiction to 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 among vectors $\vec{x}$ and $\vec{p}$ (see [17]), it is hard to say more in general about the effects of 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 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 $\frac{1}{2}\sum_j |p_j - p'_j|$, or, 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 $p'_j \le p_j$. 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 $f'(\vec{v}_0) = j$, it means that $x'_j\, g(\vec{v}_0, j) = \max_k\ x'_k\, g(\vec{v}_0, k)$ and therefore

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

Thus, $x_j\, g(\vec{v}_0, j) = \max_k\ x_k\, g(\vec{v}_0, k)$ and $f(\vec{v}_0) = j$, which is a contradiction to 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 $p'_j \ge p_j$. 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 $f(\vec{v}_0) = j$, it means that $x_j\, g(\vec{v}_0, j) = \max_k\ x_k\, g(\vec{v}_0, k)$ and therefore

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

Thus $x'_j\, g(\vec{v}_0, j) = \max_k\ x'_k\, g(\vec{v}_0, k)$ and $f'(\vec{v}_0) = j$, which is a contradiction to 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 were 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 re-mappings, 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 $\bar{\rho}(t) \le 1$. 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

$$x_j(t) := c(t)\, x_j(t - \Delta t), \quad \bar{\rho}_j(t) > \epsilon_\rho(t), \tag{27}$$
$$x_j(t) := x_j(t - \Delta t), \quad \bar{\rho}_j(t) \le \epsilon_\rho(t). \tag{28}$$

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 under-utilized, the presented adaptation *lowers the weights for the exceedingly* (with respect to a threshold) *over-utilized processors*, whereas weights for others remain unchanged. Conversely, if the system in total is over-utilized, the adaptation *raises the weights for the exceedingly* (with respect to a threshold) *under-utilized processors*. The lowering or raising of weights is carried out proportionally, either to the minimal utilization $\bar{\rho}_j(t)$ which exceeds the threshold $\epsilon_\rho(t)$, or to the maximal utilization $\bar{\rho}_j(t)$ 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 in the case of more processors.

V. Numerical Results

A. Trace Driven Simulations

We have used the MATLAB v.5 environment on an IBM RS 6000 machine to simulate a model of a router with multiple NPUs and line cards.

For router input, we have used pre-generated traffic traces. Each trace corresponds to network traffic received at one line card. The parameters for generating the traces were approximated from OC-3 traces statistics gathered in [1], [15] and [22] 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 from 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 inter-arrival times would have to be taken into account as well.

The following parameters characterize the traces:

• *Number of flows existent in a time interval*–a discrete time homogeneous Markov chain, attaining values in the interval of [8000, 240000] with uniform transition probability to states within a neighboring interval, the step of change limited within $[-5500, 5500]$ flows of difference at each iteration every 15 ms.
• *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. The direction of change (increase or decrease) in the number of packets is correlated with the direction of change in the number of flows (number of packets grows when number of packets grows and vice versa), as shown in [22] to hold.
• *Flow length*–the amount of packets in a flow. Based on [22] and on the analysis of [1], we have used exponential distribution with mean 4 to generate the individual flow lengths.
• *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 [15]. 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 [15]. Thus, the identifier vectors of flows are generated with a truncated normal distribution $\mathcal{N}(0, 1)$ out of the 32-bit integer space.

In our simulations we have simplified the problem by having each packet require an identical, constant amount of processing at the NPU. The unit of load at each NPU is equal to 1 packet. 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 μ_j = 1466 packets per ms. Note that, in reality, the amount of processing units a packet would consume would most likely vary. For example, in the case of a prefix lookup using a tree search, the tree entries would be located at various tree depths. Simulating such a load distribution would require to match the traces against a correspondent lookup table.

The low-pass filter constant r is set to $r = 3$ and the hysteresis bound ϵ_h to $\epsilon_h = 0.01$. Evaluations of the adaptation trigger are carried out at a time interval $\Delta t = 1$ ms. The number of links n and processors m in the simulations have been chosen such that the total system utilization remains close to 1, where it makes sense to investigate the performance with respect to the acceptable load-sharing bounds.

B. Load Sharing

Figures 4 and 5 show the effects of load sharing in general. A load of $n = 26$ links is processed by a router equipped with $m = 16$ 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-adapted mapping f and to load sharing with dynamically adapted mapping $f(t)$.

Figure 4 depicts the maximum and minimum processor utilization using each of the schemes, as well as the total system utilization $\rho(t)$, which is the same in all three cases. Clearly,

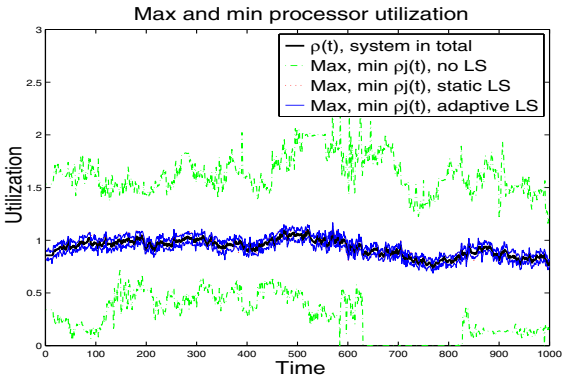

Fig. 4. Processor utilization, n=26, m=16.

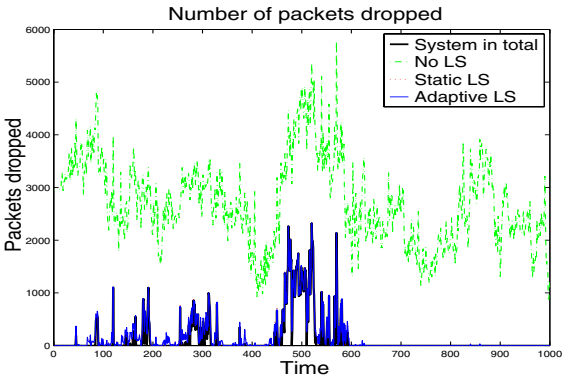

Fig. 5. Number of packets dropped, n=26, m=16.

individual processor utilization remains within close vicinity of the total system utilization when load sharing is deployed.

Figure 5 shows the amount of packets dropped under the three scenarios (assuming that as many packets have to be dropped, as exceed the processor capacity). Again, the load-sharing cases closely follow the total load curve, which represents the global minimum.

The traces used consisted of such identifier vectors that led to relatively balanced distribution of traffic to processors in the static case and therefore the small difference between the static and dynamically adapted load sharing.

C. Adaptation

Figures 6 and 7 show the benefit of dynamic adjustment. In this simulation, special traces with identifier vector patterns that result in bias towards one processor have been used. The identifier vectors in these traces have been generated such that all the traffic is mapped to one processor by the static mapping, thus being the worst case for the static mapping. The router load alternates between the biased and non-biased (same as in Section V-B) traces using weights among the two sets of traces.

In this experiment, a load of $n = 10$ links is processed by a router equipped with $m = 6$ processors.

Again, Figure 6 depicts the maximum and minimum processor utilization using each of the schemes, as well as the total system utilization $\rho(t)$. We observe that processor utilization stays within close vicinity of total system utilization in the case of the dynamically adapted load sharing, whereas it oscillates

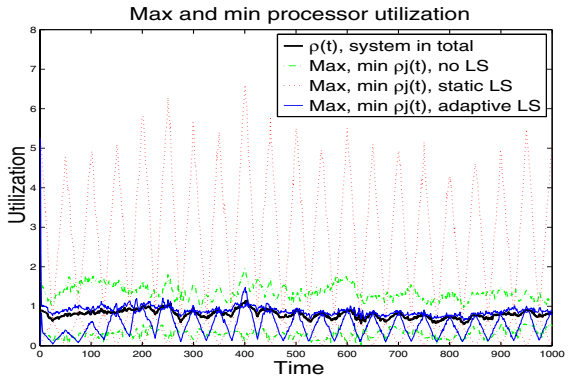

Fig. 6. Processor utilization, n=10, m=6.

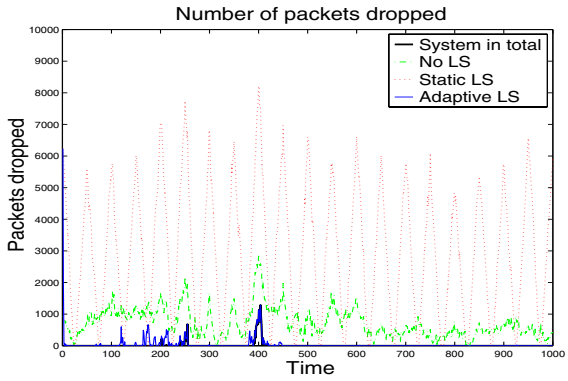

Fig. 7. Number of packets dropped, n=10, m=6.

according to which traces are input to the system in the case of the static system.

Figure 7 shows the amount of packets dropped under the three scenarios (assuming that as many packets have to be dropped, as exceed the processor capacity). Again, the adaptive case closely follows the total load curve, which represents a global minimum, whereas the static load sharing exhibits a large number of packets dropped in response to the overload of the single processor (the target of the "worst-case" traces).

D. Optimality

Figure 8 and Table I show the number of persistent flows and the number of flows that are re-mapped in the iterations of the adaptation. The results are obtained from the same simulation as in Section V-C. As the adjustment of the mapping possesses the *minimum disruption property*, only a small fraction of the request object space, and, consequently, of the persistent flow identifier vectors, is re-mapped.

The minimal amount of flow-to-processor re-mappings needed to keep the system within the bounds of acceptable load sharing can indeed be determined ex-post by solving the linear optimization problem in Def. 3 using the knowledge of flow identifier vectors $\vec{v}$ that have arrived during the intermediate time interval $(t - \Delta t, t)$. We consider the mapping $f(t)$ within the time interval $(t - \Delta t, t)$ to be unknown. In Def. 3 of the optimization problem, the mapping $f(t)$ is represented by the term $1_{\{f(t)(\vec{v})=j\}}$, which we denote as an unknown $y_{\vec{v},j}$. As $1_{\{f(t)(\vec{v})=j\}} \in \{0, 1\}$, the optimization is an integer linear programming problem, which is known to be NP-complete. However, we can attempt to relax the problem by bounding our unknown $y_{\vec{v},j} \in [0, 1]$. The solution of such optimization can be interpreted rather as a probability of $\vec{v}$ being mapped to node j by $f(t)$.

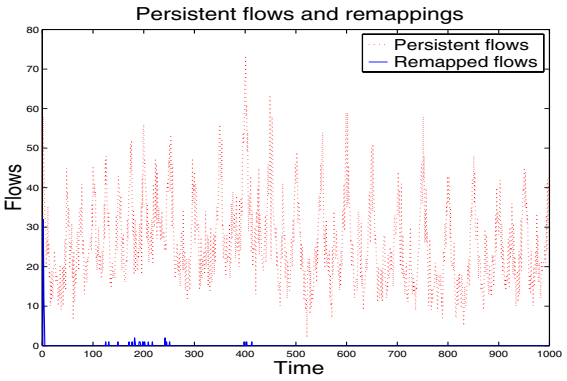

Fig. 8. Number of flows, persistent and remapped, n=10, m=6.

TABLE I
NUMBER OF FLOWS, ALL, PERSISTENT AND REMAPPED

	All	Persistent	Remapped
Simulation total			
# of flows	7'022'982	25'896	74
% of all	100.00	0.37	0.0011
% of persistent	-	100.00	0.29
Per iteration			
Max, # of flows	10'062	73	32
Max, % of all	100.00	0.82	0.40
Max, % of pers.	-	100.00	55.17

In all the iterations of the presented simulation, the relaxed linear optimization problem yielded a solution that would require *0 flow re-mappings*, showing that there is potentially room for improvement in the heuristics. However, the actual amount of re-mappings carried out by the adaptation remains close to the optimal, zero re-mapping, solution.

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 9.

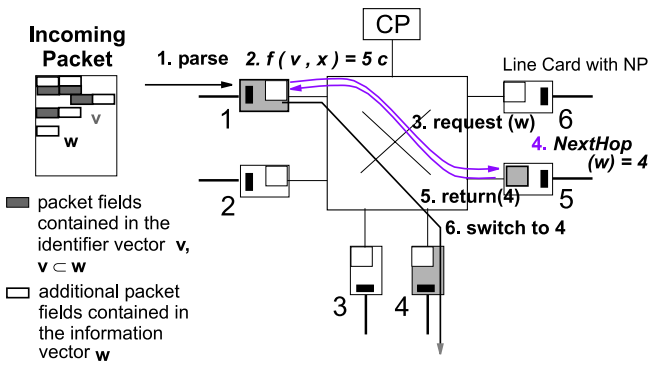

Fig. 9. Load sharing within a distributed multiprotocol router.

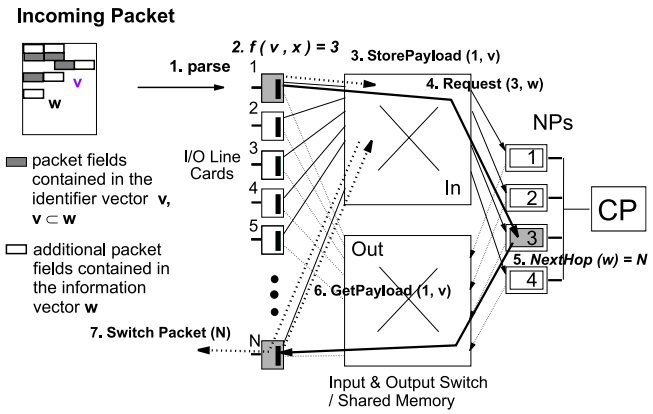

Fig. 10. Multiprotocol router, consisting of an input- and output switch/shared memory and multiple network processors, sharing the load of N line cards.

Another potentially fruitful implementation is shown in Figure 10. It is a load-sharing extension of the concepts of the highly successful products of Juniper Networks [18], 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, while the payload is temporarily stored in a distributed shared memory coupled to the input switch. Sharing the load among the NPUs would again bring 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).

A good candidate seems to be the hash function based on the Fibonacci golden ratio multiplier $\phi^{-1} = (\sqrt{5}-1)/2$, presented in [11]. The Fibonacci hash function leads to the "most random" scrambling of sequences [11]. It is defined as follows:

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

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

We have used Fibonacci hashing to compute g in our experiments.

C. Load Indicator

Another open implementation issue is how to actually measure the load of the processors or the amount of processing units spent per time interval. A good measure can be the number of memory accesses or of processing cycles an NPU has performed during the time interval Δt. A counter value is then periodically accessed by the CP.

VII. Conclusions and Future Work

We have proposed a scheme for sharing the packet processing tasks over 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 [17] and [21]. 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 utilization.

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 amount of flows to be re-mapped. The amount is very close to the minimal amount possible, as shown by comparison with a solution of the corresponding relaxed linear optimization problem. Thus the probability of packet reordering within a flow is kept low.

Such a scheme is particularly useful in routers with many input ports, 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.

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 an NPU load sharing on QoS guarantees provided by a router. These topics are currently under investigation.

References

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

[2] A. Asthana, C. Delph, H. V. Jagadish, P. Krzyzanowski. *Towards a Gigabit IP router.* Journal of High Speed Networks, Vol. 1, No. 4, pp. 281-288, 1992.

[3] G. Barish, K. Obraczka. *World Wide Web caching: trends and techniques* IEEE Communications Magazine, Vol. 38, No. 5, pp. 178-184, May 2000.

[4] T. L. Casavant and J. G. Kuhl *A taxonomy of scheduling in general-purpose distributed computing systems.* IEEE Transactions on Software Engineering, Vol. 14, No. 2, pp. 141-154, February 1988.

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

[6] H. C. B. Chan, H. M. Alnuweiri, V. C. M. Leung. *A framework for optimizing the cost and performance of next-generation IP routers.* IEEE Journal on Selected Areas in Communications, Vol. 17, No. 6, pp. 1013-1029, June 1999.

[7] D. L. Eager, E. D. Lazowska, J. Zahorjan. *Adaptive load sharing in homogenous distributed systems.* IEEE Transactions on Software Engineering, Vol. SE-12, No. 5, pp. 662-675, May 1986.

[8] H. El-Rewini, H. H. Ali and T. Lewis. *Task scheduling in multiprocessing systems.* IEEE Computer, Vol. 28, No. 12, pp. 27-37, December 1995.

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

[10] G. Goldszmidt and G. Hunt. *Scaling Internet services by dynamic allocation of connections.* Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management, pp. 171-184, 24-28 May 1999.

[11] D. E. Knuth. *The art of computer programming, Vol. 3, Sorting and searching.* Addison-Wesley, 1973.

[12] O. G. Koufopavlou, A. N. Tantawy, M. Zitterbart. *Analysis of TCP/IP for High Performance Parallel Implementations.* 17th IEEE Conference on Local Computer Networks, Minneapolis, September 1992.

[13] V. P. Kumar, T. V. Lakshman, D. Stilliadis. *Beyond best effort: router architectures for the differentiated services of tommorrow's Internet*, IEEE Communications Magazine, pp. 152-164, May 1998.
[14] T. Kunz. *The influence of different workload descriptions on a heuristic load balancing scheme.* IEEE Transactions on Software Engineering, Vol. 17, No. 7, pp. 725-730, July 1991.
[15] National Laboratory for Applied Network Research (NLANR). *WAN traffic distribution by address size, Fix-West trace*, http://www.nlanr.net/NA/Learn/Class, May 1997.
[16] C. Partridge et al. *A fifty gigabit per second IP router*, IEEE/ACM Transactions on Networking 6, 3, pp. 237-248, June 1998.
[17] K. W. Ross. *Hash routing for collections of shared web caches.* IEEE Network, Vol. 11, No. 6, November-December 1997.
[18] Ch. Semeria. *Internet backbone routers and evolving Internet design*, Juniper Networks white paper, http://www.juniper.net, September 1999.
[19] B. A. Shirazi, A. R. Hurson and K. M. Kavi, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE CS Press, 1995.
[20] A. Tantawy, M. Zitterbart. *Multiprocessing in high performance IP routers.* Proceedings of the 3rd IFIP WG 6.1/6.4 Workshop on Protocols for High Speed Networks, Stockholm, Sweden, May 1992.
[21] D. G. Thaler, C. V. Ravishankar. *Using name-based mappings to increase hit rates.* IEEE/ACM Transactions on Networking, Vol. 6, No. 1, pp. 1-14, February 1998.
[22] K. Thompson, G. J. Miller, R. Wilder. *Wide-area Internet traffic patterns and characteristics.* IEEE Network, Vol. 11, No. 6, pp. 10-27, November-December 1997.
[23] H. Zhu, T. Yang, Q. Zheng, D. Watson, O. H. Ibarra, T. Smith *Adaptive load sharing for clustered digital library servers.* Proceedings of the Seventh International Symposium on High Performance Distributed Computing, pp. 235-242, July 1998.