

**On Achieving Fairness in the
Joint Allocation of
Processing and Bandwidth Resources:
Principles and Algorithms**

Yunkai Zhou
and
Harish Sethu

Technical Report DU-CS-03-02
Department of Computer Science
Drexel University
Philadelphia, PA 19104
July 2003

On Achieving Fairness in the Joint Allocation of Processing and Bandwidth Resources: Principles and Algorithms

Yunkai Zhou and Harish Sethu

Abstract—The problem of achieving fairness in the allocation of the bandwidth resource on a link shared by multiple flows of traffic has been extensively researched over the last decade. However, with the increasing pervasiveness of optical networking and the occasional trend toward using over-provisioning as the solution to bandwidth congestion, a router’s processor also becomes a critical resource to which, ideally speaking, all competing flows should have fair access. For example, if the network is not fair in allocating processing resources, denial of service attacks based on an excessive use of the router processor (such as by using unnecessary optional headers) become possible. In this report, we investigate the issue of achieving fairness in the joint allocation of the processing and bandwidth resources. We first present a simple but powerful general principle for defining fairness in such systems based on any of the classic notions of fairness such as max-min fairness, proportional fairness and utility max-min fairness defined for a single resource. We apply our principle to a system with a shared processor and a shared link with max-min fairness as the desired goal. We then propose a practical and provably fair packet-by-packet algorithm for the joint allocation of processing and bandwidth resources. We demonstrate the fairness achieved by our algorithm through simulation results using both synthetic and real gateway traffic traces. The principles and the algorithm detailed in this report may also be applied in the allocation of other kinds of resources such as power, a critical resource in mobile systems.

Index Terms—Fairness, resource allocation, processor sharing, max-min.

I. INTRODUCTION

A. Introduction and Motivation

Fairness in the allocation of resources in a network shared amongst multiple users is not only an intuitively desirable goal but also one with many practical benefits. Fairness in traffic management can improve flow and user isolation, offer a more predictable performance, and eliminate certain kinds of bottlenecks. In addition, strategies and algorithms for fair management of network traffic can serve as a critical component of Quality-of-Service (QoS) mechanisms to achieve certain guaranteed services such as delay bounds and minimum bandwidths. Fair resource allocation strategies can also help in countering certain kinds of denial-of-service (DoS) attacks [1]. Various formal notions of fairness have been proposed in the literature to precisely define what is fair in the allocation of a resource amongst competing flows. These include, among

others, max-min fairness [2–5], proportional fairness [6], and utility max-min fairness [7].

Based on these notions of fairness—most commonly, based on the notion of max-min fairness—much research over the last decade or two has focused on the allocation of the bandwidth resource on a link [3, 4, 8–12]. It has also been shown that concepts and algorithms for achieving fairness in the allocation of a single resource can be extended to the case with multiple resources *of the same kind* [13]. However, as flows of traffic traverse a computer network, they share many different kinds of resources such as link bandwidth, buffer space, time on the router processors and also electrical power, a critical resource in mobile systems. The ultimate goal, therefore, should be the overall fairness in the *joint* allocation of all resources shared by the flows of traffic and not just one specific kind of resource such as the link bandwidth. For example, if the network is not fair in allocating processing resources, DoS attacks based on an excessive use of the router processor (such as by using unnecessary optional headers) become possible.

The significance of considering the fair allocation of more than just the link bandwidth is increasingly becoming apparent today, since the link bandwidth is often not the only critical resource. With the current pervasiveness of optical networking in the Internet backbone, and with the occasional trend toward using over-provisioning as the solution to congestion in the edge networks, a router’s processor is often also a critical resource to which, ideally speaking, all competing flows should have fair access. Given the fact that processing requirements of different packets vary widely, the issue of fairness in the allocation of the processing resources gains significance. In addition, besides the fact that packet lengths can vary widely, the presence of optional headers and the various kinds of control information carried by packets create a wide variation in the ratio of a packet’s demand for bandwidth and its demand for processing cycles. Thus, packets of the same length cannot be guaranteed to have similar requirements for the processing resources on a router. In fact, the processing delay plotted as a function of the packet length shows that the processing requirements of packets vary across a wide range even for packets of the same length [14]. Thus, one cannot achieve overall fairness merely with the fair allocation of link bandwidth alone, or merely through the fair allocation of the processing resource alone, since different flows—and different packets within the same flow—may have very different demands for these two kinds of resources. All of this begs the question this report seeks to address: how does

This work was supported in part by NSF CAREER Award CCR-9984161 and U.S. Air Force Contract F30602-00-2-0501. A preliminary version of this research appeared in *Proc. Int’l Workshop on Quality of Service (IWQoS)*, Monterey, CA, June 2003.

one achieve fairness in the *joint* allocation of the processing and bandwidth resources?

The need for fairness in the joint allocation of multiple heterogeneous resources has also been recognized in other contexts besides the one discussed here. For example, it has been recognized that fair allocation of both the channel bandwidth and the power consumed needs to be achieved simultaneously in mobile networks where power and bandwidth are both critically important and scarce resources [15]. However, a rigorous theoretical framework that may be universally employed as a guide in the design of practical algorithmic strategies for the joint allocation of such heterogeneous sets of resources does not exist.

In this technical report, we investigate the issue of fairness in such systems and develop a general principle that forms the foundation for the design of practical and fair strategies for use in routers. We also present an evaluation of the practical strategies proposed in this report using both synthetic and real gateway traffic traces.

B. Contributions

In the joint allocation of the processing and bandwidth resources, if a certain resource is never the bottleneck, then the fair allocation strategy degenerates to the fair allocation of just the other resource. For example, if the available bandwidth is large enough that no flow experiences congestion due to lack of bandwidth alone, one only needs to worry about the allocation of the processing resource. Fair allocation of a single bottleneck resource has been studied extensively in the literature and has led to a large number of practical algorithms that are in use today in Internet routers, operating systems, and transport-level protocols. This report, on the other hand, answers the question of what is a fair allocation when more than one resource is congested and extends the notions of fairness applied to a single resource to systems with multiple heterogeneous resources.

We define an *essential* resource as one for which a flow's demand does not reduce with an increase in the allocation of other resources to the flow. A number of resources such as the link bandwidth, processor or power, in most contexts, are essential resources. On the other hand, buffer resources in a network are often non-essential resources as per the above definition; for example, in a system with a buffer and a link, a flow uses the buffer only if the link resource is currently unavailable to it, and thus a flow's demand for the buffer resource reduces as more of the link bandwidth is allocated to it. Note that a non-essential resource does not necessarily mean that it is not useful. In the system model used in this report, we assume that the flows are in competition for resources that are all essential. The issue of achieving fairness in a system where flows have to compete for a non-essential resource such as a buffer entails a different set of challenges than those considered here, and is addressed in some other recent works such as [16].

We define a pair of resources as *related* to each other if a flow's demand for one resource uniquely determines its demand for the other resource. Resources in a set are said to be *related* if each resource is related to every other resource in the set. Resources in real scenarios are almost always related since the demands of a flow for different individual resources are often

related to each other. For example, since each packet is associated with certain processing and bandwidth requirements, a specific increase in a flow's demand for link bandwidth is typically associated with a specific increase in its demand for the processing resource. A simpler example, involving multiple resources of the same kind, is a tandem network with multiple links where the demand of a flow for bandwidth is the same on all the links. In the system model used in this report, we assume multiple resources that are related, although we make no assumptions on the specific nature of the relationship between a flow's demand for different resources. The existence of a relationship between the demands of a flow for various resources calls for the *joint* allocation of these resources, as opposed to an independent and separate allocation of the resources.

The primary contribution of this report is a theoretical framework based on which one can define fairness in the joint allocation of multiple heterogeneous resources that are essential and related. We make no assumptions on the notion of fairness; in fact, our framework may be applied to any of several notions of fairness such as max-min fairness, proportional fairness or utility max-min. Through illustrative examples, we claim that, at each instant of time, it is the maximum of a flow's normalized demand for the various resources that should count in the decisions made by a fair resource allocation algorithm. We then develop the fundamental principles of fairness for systems with multiple essential and related heterogeneous resources, and propose the *Principle of Fair Essential Resource Allocation* or the *FERA principle*, expressed within a rigorous theoretical framework. We also prove that, under certain conditions, there exists a unique, fair, and work-conserving resource allocation policy which satisfies the FERA principle.

Given the FERA principle, we proceed to apply it to a system with a shared processor and a shared link, using max-min fairness as the notion of fairness. We propose an ideally fair policy, called the *Fluid-flow Processor and Link Sharing (FPLS)* algorithm, for the joint allocation of processing and bandwidth resources. We then develop a practical and *provably* fair packet-by-packet approximation of the FPLS algorithm, called *Packet-by-packet Processor and Link Sharing (PPLS)*. The PPLS algorithm, based on an extension of the Deficit Round Robin algorithm [10], has a per-packet work complexity of $O(1)$. We illustrate the fairness of the PPLS algorithm using both synthetic traffic and real gateway traffic traces.

Even though this report primarily focuses on processing and bandwidth resources, the FERA principle may be readily applied to a variety of contexts beyond those discussed in this report.

C. Organization

The rest of this report is organized as follows. Section II introduces a generic notation to represent notions of fairness. This section also describes the general system model with multiple shared resources considered in this study, along with our notation. Section III presents the Principle of Fair Essential Resource Allocation for the system model under consideration. Section IV applies the FERA principle to a system with a shared processor and a shared link, and proposes a practical and fair scheduling algorithm for the joint allocation of the processing

and bandwidth resources, called the Packet-by-packet Processor and Link Sharing (PPLS) policy. The fairness properties of the PPLS strategy are demonstrated by simulation experiments using both synthetic and real gateway traffic in Section V. Finally, Section VI concludes the report.

II. SYSTEM MODEL AND NOTATION

A. Generic Notation for Notions of Fairness

Consider a set of N flows, $1 \leq i \leq N$, competing for a single shared resource which may be consumed at a peak rate of R . Denote by w_i the weight of flow i , indicating the flow's relative rightful share of the resources. For a flow under a Differentiated Services (DiffServ) framework [17], its weight is determined by its traffic class among the 64 possible classes; for a flow in a best-effort network, its weight is typically the same as that of all other flows.

Several different notions of fairness have been proposed in the research literature for the allocation of a single shared resource among a set of requesting entities. All of these notions specify a particular rate of consumption of the resource for each of the flows, given the consumption rate demanded by the flows. In this subsection, we develop a generic notation that can express any of these notions of fairness.

Without loss of generality, we assume that the entities competing for the single shared resource are traffic flows. Let d_i be the demand of flow i for the shared resource. Define the normalized demand of flow i , D_i , for the resource as follows:

$$D_i = \frac{d_i}{R}.$$

The normalized demand of flow i indicates the fractional share of the resource demanded by the flow. Denote by a_i the allocated resource consumption rate for flow i . Define the normalized allocation of flow i , A_i , as follows:

$$A_i = \frac{a_i}{R}.$$

The normalized allocation of flow i indicates the fractional share of the resource allocated to flow i . Any notion of fairness, thus, specifies how to distribute the fractional share of the resource allocated to each flow, given the desired share of this resource.

For the sake of convenience, throughout this report we use vectors to indicate values corresponding to a set of flows. We denote a vector by the indexed value in a pair of square brackets. For instance, we denote the vector of normalized demands as $[D_i]$.

Therefore, given the normalized demand vector $[D_i]$ and the weight vector $[w_i]$, any given notion of fairness may be represented as

$$[A_i] = \mathcal{F}(C, [D_i], [w_i]) \quad (1)$$

where C is the constraint, described later in greater detail, imposed on the system. The function \mathcal{F} is different for different notions of fairness such as max-min fairness, proportional fairness or utility max-min fairness. Note that the notion of fairness in (1) imposes no dimension on any variable, thus making it applicable to systems with multiple heterogeneous resources.

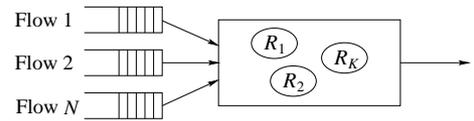


Fig. 1. A general system model.

Often, a fairness notion implies a certain way to compute the utility to a flow of the allocations. Different notions of fairness assume different utility functions, though all are non-decreasing functions with respect to quantity of the allocated resource. For example, max-min fairness implies a linear utility function, proportional fairness uses a logarithmic utility function, and in utility max-min, each flow determines its own utility function. The notion of fairness in (1) represents a general notation to describe how, given a certain vector of demands, one may determine the allocated consumption rate of the resource for each flow, in order that the utilities corresponding to the allocations satisfy the given fairness notion with respect to the demands for utility. In other words, the notation of (1) implicitly incorporates utility functions into the notion of fairness.

The constraint C is used as a parameter in the function \mathcal{F} because, given the same demand and weight vector, the fair allocation is different under different constraints imposed on the system. The constraint C can be used to incorporate the performance level achieved by the allocation. For example, an allocation of no resource to any flow may also be considered a fair allocation by the max-min fair criterion albeit one that leads to very poor performance. In general, this parameter allows us to define the fairness of non-work-conserving allocation strategies by not imposing a specific level of performance achieved by the allocation in the definition of fairness. As a simple example, the constraint C can be just the sum of the utilities achieved by all flows.

Note that, in the research literature, notions of fairness have not been defined for multiple heterogeneous resources¹. We use the above notation that specifies a notion of fairness for a single resource and extend the notion to multiple heterogeneous resources in subsequent sections.

B. System Model and Assumptions

In our system model, a set of N flows, $1 \leq i \leq N$, compete for a set of K related and essential resources, $1 \leq j \leq K$, as shown in Fig. 1. As also described in Section I-B, we define an essential resource as one for which a flow's demand does not reduce with an increase in the allocation of other resources to it. Since a buffer is often not an essential resource, our assumption that flows only compete for essential resources implies that if there are buffers in the network shared by the flows, these buffers are of infinite capacity so that the flows never compete

¹Some notions of fairness such as max-min fairness and proportional fairness can be defined for multiple resources of the same kind (e.g., a network of links), under the assumption that, if a flow receives allocation of several resources, the allocation of each of these resources it receives is identical [2, 6]. However, it is not straightforward to extend these notions of fairness to systems with multiple heterogeneous resources. On the other hand, it can be readily verified that our framework is the same as these notions of fairness if the shared resources are of the same kind.

for the buffer resource. In developing our fundamental principles of fairness, we make no assumptions on the specific actions of the scheduler or the specific order in which the packets use the K resources.

Note that in this general model, we also make no assumptions on the internal architecture of the set of shared resources. It can be a simple sequence of resources such as in a tandem network with multiple links, a parallel structure such as the resources of electric power and bandwidth in a wireless sensor network, or a more complex hybrid.

Denote by R_j the peak rate at which resource j may be consumed. For example, in the case of a link resource L , R_L is the peak bandwidth available on the link. As before, denote by w_i the weight of flow i . Let $d_{i,j}$ be the consumption rate demanded by flow i for the shared resource j . Our assumption of related resources implies that, given $d_{i,k}$, one can determine $d_{i,j}$ for all $j \neq k$. Denote by $a_{i,j}^q$, the consumption rate of the shared resource j allocated to flow i under the allocation policy q .

III. THE FERA PRINCIPLE

A. The Concept of the Prime Resource

We begin with a few preliminary definitions.

Definition 1: Define the normalized demand of flow i for resource j , $D_{i,j}$, as follows:

$$D_{i,j} = \frac{d_{i,j}}{R_j}.$$

Define the *largest normalized demand* of flow i , \mathcal{D}_i , as the maximum amongst the normalized demands of flow i for all resources. That is,

$$\mathcal{D}_i = \max_j \{D_{i,j}\}.$$

Definition 2: Define $A_{i,j}^q$ as the normalized allocation of resource j to flow i under allocation policy q , i.e.,

$$A_{i,j}^q = \frac{a_{i,j}^q}{R_j}.$$

Definition 3: The *largest normalized allocation* of a flow i under allocation policy q , denoted by \mathcal{A}_i^q , is defined as the maximum amongst the normalized allocations to flow i of all resources. That is,

$$\mathcal{A}_i^q = \max_j \{A_{i,j}^q\}.$$

Definition 4: Under an allocation policy q , a resource is said to be a *prime resource* of flow i , denoted by \mathcal{B}_i^q , if and only if, the normalized allocation of this resource to flow i is the largest normalized allocation. In other words,

$$\mathcal{B}_i^q = \arg \max_j \{A_{i,j}^q\} = \arg \max_j \left\{ \frac{a_{i,j}^q}{R_j} \right\}$$

where $\arg \max_x f(x)$ indicates the value of the argument x corresponding to the maximum value of function $f(x)$. In other words, we have

$$A_{i,\mathcal{B}_i^q}^q = \max_j \{A_{i,j}^q\} = \mathcal{A}_i^q.$$

TABLE I

EXAMPLES ILLUSTRATING WHAT IS A FAIR ALLOCATION IN A SYSTEM WITH A SHARED PROCESSOR P AND A SHARED LINK L . IN ALL OF THESE EXAMPLES, THE TOTAL AMOUNT OF RESOURCE P IS 100 MHz, AND THE TOTAL AMOUNT OF RESOURCE L IS 100 MBPS.

	Flow ID	Demand		Allocation	
		P (MHz)	L (Mbps)	P (MHz)	L (Mbps)
A	1	75	25	75	25
	2	25	75	25	75
B	1	225	75	75	25
	2	50	150	25	75
C	1	100	20	50	10
	2	100	10	50	5

In networking terminology, a bottleneck resource is one that is the most congested. It is critical to note that neither the resource for which a flow has the largest normalized demand nor its prime resource under an allocation policy is necessarily the same as the bottleneck resource in the system.

Note that a flow may have more than one prime resource. The prime resource is defined based on the actual allocations and not on the demand of the flows for the resources.

B. The FERA Principle

We introduce our principle with a few illustrative examples shown in Table I. In these examples, two flows with equal weights, labeled as 1 and 2, share two different resources: a processor P for packet processing, and a link L for packet transmission. The system model in these examples is the same as the one we will discuss later in Fig. 2. The peak processing rate is 100 million processor cycles per second, i.e., 100 MHz, and the peak link rate is 100 Mbps. Let us assume linear utility functions and max-min as the notion of fairness. In addition, for the sake of convenience, we also assume in these examples a proportional relationship between a flow's demands for these resources, and therefore, a proportional relationship between the allocations. In other words, the ratio of a flow's demand for one resource and its demand for another resource is always a constant.

In example A, assume that packets in flow 1 are all small, and therefore, its demand for bandwidth is small relative to its demand for processing time. In contrast, assume that packets in flow 2 are large, and therefore, its demand for bandwidth is large relative to its demand for processing time. To better illustrate the concept, we exaggerate the difference between their demands as follows: flow 1 has a demand of 75 MHz for processing time and 25 Mbps for bandwidth, while flow 2's demands are, respectively, 25 MHz and 75 Mbps. If a work-conserving allocation policy is used, there is enough of both resources to satisfy the demands of both the flows and so the allocations are exactly the same as the demands for each of the resources. Note that for flow 1, the prime resource is P , while for flow 2, it is L .

Next, consider what happens when both flows proportionally increase their demands for both resources. In example B, in comparison to example A, flow 1 increases its demands by a factor of three while flow 2 doubles its demands. Specifically, the demands for flow 1 become 225 MHz for P and 75 Mbps for L , while those for flow 2 become 50 MHz and 150 Mbps, respectively. A fundamental principle behind the max-min notion of fairness is that, given no additional resources, a flow should not be able to increase its allocation by merely demanding more. Thus, the fair allocation should be the same as in example A, as shown in example B. Again, the prime resource for either flow remains the same as in the previous example.

We discuss example B further. Obviously, in this case, neither flow is satisfied by the allocated resources. Is the allocation actually fair?

One might argue that both flows should get equal bandwidth from a fair allocation, since ultimately both flows will depart from this system and the processor is only an intermediate resource before the flow's packets reach the link resource. Based on this notion, we can compute the allocations as

$$\begin{cases} 3x + x/3 & \leq 100 \\ 2x & \leq 100 \end{cases}$$

where x is the bandwidth allocated to either flow, in units of Mbps. It can be readily verified that, under a work-conserving allocation policy, flow 1 gets 90 MHz of processing time and flow 2 gets only 10 MHz, while both flows get 30 Mbps of bandwidth. While this allocation underutilizes the link resources, that is not an argument against its fairness. The unfairness arises from the fact that it unnecessarily favors the flow whose prime resource is the "intermediate" resource, which turns out to be flow 1 in this case. Another argument against this notion is that, even though it is true that the processor in this case is positioned ahead of the link, it does not necessarily mean that the processing resource becomes less important, or less preferred, as compared to the link, which is positioned as the "final" resource.

Another allocation philosophy may be to allocate resources based on a fair allocation of the most congested resource as measured by the sum of the normalized demands for the resource. In this example, the processing resource P is the most congested resource. One may allocate resource P fairly as

$$\begin{cases} 2y & \leq 100 \\ y/3 + 3y & \leq 100 \end{cases}$$

where y is the processing resources allocated to either flow, in units of MHz. Under a work-conserving allocation policy, flow 2 gets 90 Mbps of bandwidth and flow 1 gets only 10 Mbps, while both flows get 30 MHz of processing resources. Note that this allocation philosophy has a similar weakness as the one based on the fair allocation of the link resource. It unnecessarily favors the flow whose largest normalized demand is not for the most congested resource.

One may suggest the following slight modification to the allocation strategy: to fairly allocate the most congested resource as measured by the sum of the normalized *allocations* for the resource. However, it can be shown that such an allocation may

not exist. Assume a certain resource r is the most congested resource. Let α denote the flow with the smaller demand for resource r and let β denote the other flow. Assume that the normalized allocations of resource r are z_α and z_β for the two flows. It can be verified that the normalized allocations of the other resource are $3z_\alpha$ and $z_\beta/3$, independent of whether the resource r is the processing resource P or the bandwidth resource L . Since resource r is the most congested resource as measured by the sum of the normalized allocations, we have,

$$3z_\alpha + z_\beta/3 \leq z_\alpha + z_\beta$$

which leads to $3z_\alpha \leq z_\beta$. Since both flow have a high demand, under the max-min notion, this condition cannot lead to a fair allocation except for the trivial case where $z_\alpha = z_\beta = 0$. Thus, it may not be possible to achieve a fair allocation of the most congested resource as measured by the sum of the normalized allocations of the resource.

Based on the discussions above, we claim that in a network where no explicit preference of one resource over another exists (i.e., each resource is essential), fairness should not be defined based only on a single resource, no matter how this single resource is determined, and whether it is determined before allocation (i.e., based on demand) or after allocation (i.e., based on allocation). Instead, the fairness in such a system should be defined with overall consideration of various resources involved in the system and the relationships between the demands for the various resources.

Given this observation, one may propose yet another scheme to define fairness: the sum of the normalized allocations of the resources computed for each flow should be max-min fair. In the previous example B, this leads to an allocation of 75 MHz of processing time and 25 Mbps of bandwidth for flow 1, and 25 MHz of processing time and 75 Mbps of bandwidth for flow 2. In this case, for both flows, the sum of the normalized allocations of the two resources is $75/100 + 25/100 = 1$. While this appears to be a reasonable strategy for fair allocation, this scheme of fairness cannot, in fact, be extended to other situations. This is illustrated by example C described below.

Assume that both flows have a demand of 100 MHz for resource P , while the demands for resource L are 20 Mbps and 10 Mbps for flows 1 and 2, respectively. Note that in this example, there is sufficient link bandwidth available for the demands of both flows, i.e., the flows are not in competition for resource L . In other words, the system regresses into an allocation of a single resource P . Applying the max-min notion of fairness on the single resource P , we know that the fair allocation would be 50 MHz of processing time for each flow, leading to 10 Mbps and 5 Mbps of bandwidth for flows 1 and 2, respectively. Thus, the ideally fair allocation leads to 0.6 and 0.55 as the sum of the normalized allocations. Clearly, if we were to not be max-min fair in the sum of the normalized allocations of the resources to each flow, we would not get this result. This illustrates that the strategy of achieving max-min fair distribution in the sum of the normalized allocations fails to serve as the basis to define fairness in the allocation of multiple resources.

The fair allocation strategies in the three examples do have one property in common: the largest normalized allocations of the flows are distributed in a max-min fair manner among the

flows. In our case with equal weights for the flows, the largest normalized allocations are equal for the two flows. In the first two examples in Table I, resource P is the prime resource for flow 1, while the prime resource for flow 2 is resource L . In both examples, the largest normalized allocation equals 0.9 for each flow. In the third example, the processor P is the prime resource for both flows, and this time the largest normalized allocation is 0.5 for both flows.

The observations from the above examples lead to the significance of incorporating the largest normalized allocation for each flow into a strategy for extending a notion of fairness to the allocation of multiple resources. In our examples, the fair allocation policy is to simply equalize the largest normalized allocations for different flows. In the general situation, different notions of fairness may be used and flows may have different weights, different largest normalized demands, and very different utility functions. This leads to the following *Principle of Fair Essential Resource Allocation*.

Principle of Fair Essential Resource Allocation (FERA): In a system with multiple related and essential resources, an allocation policy q is said to be fair as per the notion of fairness \mathcal{F} , if and only if, the largest normalized allocations are distributed fairly, as per the notion of fairness \mathcal{F} , with respect to the largest normalized demands. In other words, allocation policy q is fair as per \mathcal{F} if and only if,

$$[A_i^q] = \mathcal{F}(C, [D_i], [w_i])$$

where C is some constraint imposed on the system.

C. Fair Work-Conserving Allocation Policy

Recall that we make no assumption on whether or not the allocation policy is work-conserving. Therefore, under different constraints, a single system can have more than one fair allocation policy as per the same notion of fairness. Given a constraint, however, there exists a unique work-conserving fair allocation policy in most situations, as will be proved in this section.

First, we formally define a work-conserving policy in the allocation of multiple resources. Recall that in the allocation of a single resource, an allocation policy is work-conserving if and only if one of the following two situations occurs.

- 1) The demands of all flows are satisfied.
- 2) The shared resource is completely allocated.

In other words, no more of the resource can be further allocated to the flows. The same idea can be applied to the allocation of multiple resources, except that now it is possible that only one resource is fully utilized.

Definition 5: In the allocation of multiple resources, an allocation policy is said to be *work-conserving*, if and only if, upon completion of the allocation, no more of any resource can be further allocated to a flow without also reducing the amount of some resource allocated to another flow.

Next we introduce two general classes of fairness notions which describe the conditions under which the uniqueness of the fair work-conserving allocation policy will hold.

Definition 6: A notion of fairness \mathcal{F} is said to be *uniquely deterministic*, if and only if, given the constraint C , the normalized demand vector $[D_i]$, and the weight vector $[w_i]$, one

can uniquely determine the normalized allocation vector $[A_i]$ as given in (1).

Definition 7: A notion of fairness \mathcal{F} is said to be *non-decreasing*, if and only if, given the normalized demand vector $[D_i]$ and the weight vector $[w_i]$, the normalized allocation $[A_i]$ is such that, for any two different constraints C_1 and C_2 , one of the following holds true:

$$\begin{aligned} \mathcal{F}(C_1, [D_i], [w_i]) &< \mathcal{F}(C_2, [D_i], [w_i]) \\ \mathcal{F}(C_2, [D_i], [w_i]) &< \mathcal{F}(C_1, [D_i], [w_i]). \end{aligned}$$

Here $<$ is a relational operator between two vectors of identical dimensions, and $[u_i] < [v_i]$ implies $\forall i, u_i \leq v_i$. This definition of non-decreasing fairness notion can be also expressed as follows: when allocating a single resource under a non-decreasing fairness notion, no flow will get a lesser amount of the resource if the total amount of the shared resource increases.

These classes of fairness notions are actually very broad; it may be readily verified that many popular notions of fairness are both non-decreasing and uniquely deterministic. These include max-min fairness [3–5], proportional fairness [6], and utility max-min fairness [7] if the utility functions are non-decreasing.

Theorem 1: If the applied notion of fairness is both non-decreasing and uniquely deterministic, there exists a unique fair work-conserving allocation policy that satisfies the FERA principle as stated in Section III-B.

Proof: The reader is referred to Appendix I. ■

IV. FAIR JOINT ALLOCATION OF PROCESSING AND BANDWIDTH RESOURCES

In this section, we apply the framework established in the previous section into an important context of special interest: the fair joint allocation of a shared processor P and a shared link L under the max-min notion of fairness and linear utility functions.

A. System Model

In this system model, a set of N flows share a processor P and a link L , as shown in Fig. 2. Packets from each flow are processed by processor P first and then transmitted onto the output link L . Denote by R_L the peak bandwidth rate of link L and by R_P the peak processing rate of processor P . Packets of each flow await processing by the processor in an input buffer of infinite capacity, and then upon completion of the processing, await transmission on the output link in another buffer of infinite capacity. The joint allocation of the processing and bandwidth resources is accomplished by the scheduler which acts on the packets in the input buffers and appropriately orders them for processing by the processor. No scheduling action takes place after the processing; processed packets are received in the buffer between the processor and the link, and are transmitted in a first-come-first-served fashion.

Denote by w_i the weight of flow i , $1 \leq i \leq N$, indicating the flow's relative rightful share of the resources.

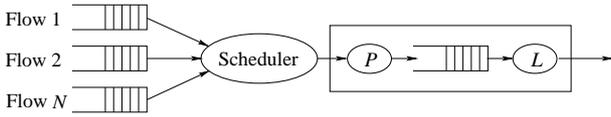


Fig. 2. The system model with a shared processor P and a shared link L .

B. Fluid-flow Processor and Link Sharing

Denote by S the system illustrated in Fig. 2. We first consider fluid-flow traffic through system S , and describe an ideally fair allocation strategy called the *Fluid-flow Processor and Link Sharing (FPLS)* algorithm. FPLS is intended to serve the same purpose for system S as that served by Generalized Processor Sharing (GPS) for a system with just a single shared link or a single shared processor [4, 5].

In GPS, it is assumed that traffic from each flow can be divided into infinitesimally small chunks, and each chunk has its demand for access to the link L depending on the size of the chunk. The GPS scheduler visits each active flow's queue in a round-robin fashion, and serves an infinitesimally small amount of data from each queue, in such a way that during any infinitesimal interval of time, it can visit each queue at least once. In our study, this assumption is still valid, and we further assume that each infinitesimal chunk also has its demand for the processing time on the shared processor P .

At each time instant τ , the prime resource for each flow, according to Definition 4, can be determined based on its instantaneous demands for processing time and bandwidth. In addition, we assume that during each infinitesimal interval of time, $[\tau, \tau + \Delta\tau)$, the prime resource for each flow does not change.

Note that in GPS, it is guaranteed that during each infinitesimal interval of time, the chunks of each flow are scheduled in such a way that, for each flow, the total demand for bandwidth corresponding to the chunks of the flow scheduled in this period is proportional to the weight of the flow. Extending GPS to our case leads to the following: under the ideally fair allocation policy for system S , it is guaranteed that, during each infinitesimal interval of time, the chunks of each flow are scheduled in such a way that, for each flow, the total *normalized* demand for its *prime resource* corresponding to the chunks of the flow scheduled in this period is proportional to the weight of the flow. We refer to this as *Fluid-flow Processor and Link Sharing (FPLS)*. It can be readily verified that the FPLS strategy meets the FERA principle described in Section III-B.

C. Packet-by-packet Processor and Link Sharing

It is apparent that FPLS is an ideally fair but unimplementable policy, in the same sense as GPS. In reality, network traffic is always packetized, and therefore, we next present a practical approximation of FPLS, called *Packet-by-packet Processor and Link Sharing (PPLS)*. The pseudo-code of PPLS is shown in Fig. 3.

The PPLS algorithm extends one of the most practical and simple scheduling strategies, Deficit Round Robin (DRR) [10], used in the allocation of bandwidth on a link. For each flow, the DRR algorithm maintains a *deficit counter (DC)*, which is

```

1  Initialize:
2  FlowList  $\leftarrow$  NULL;

3  Enqueue: /* Invoked whenever a packet arrives */
4   $p \leftarrow$  ArrivingPacket;
5   $i \leftarrow$  Flow( $p$ ); /* Flow of packet  $p$  */
6  if (ExistsInFlowList( $i$ ) = FALSE) then
7    Append flow  $i$  to FlowList;
8     $PDC_i \leftarrow 0$ ;
9     $LDC_i \leftarrow 0$ ;
10 end if;

11 Dequeue: /* Always running */
12 while (TRUE) do
13   if (FlowList  $\neq$  NULL) then
14      $i \leftarrow$  HeadOfFlowList;
15     Remove  $i$  from FlowList;
16      $PDC_i \leftarrow PDC_i + PQ_i$ ;
17      $LDC_i \leftarrow LDC_i + LQ_i$ ;
18     if ( $PDC_i > \max PDC_i$ ) then
19        $PDC_i \leftarrow \max PDC_i$ ;
20     end if;
21     if ( $LDC_i > \max LDC_i$ ) then
22        $LDC_i \leftarrow \max LDC_i$ ;
23     end if;
24     while (QueueIsEmpty( $i$ ) = FALSE) do
25        $p \leftarrow$  HeadOfLinePacketInQueue( $i$ );
26       if (Size( $p$ ) >  $LDC_i$  OR
27         ProcessingCost( $p$ ) >  $PDC_i$ ) then
28         break; /* escape from the inner while loop */
29       end if;
30        $PDC_i \leftarrow PDC_i -$  ProcessingCost( $p$ );
31        $LDC_i \leftarrow LDC_i -$  Size( $p$ );
32       Schedule  $p$ ;
33     end while;
34     if (QueueIsEmpty( $i$ ) = FALSE) then
35       Append queue  $i$  to FlowList;
36     end if;
37   end if;
38 end while;

```

Fig. 3. Pseudo-code of the Packet-by-packet Processor and Link Sharing (PPLS) algorithm.

incremented in each round by a predetermined quantity, *quantum*. When the scheduler visits one flow, it transmits the packets from this flow with a total length no more than the deficit counter associated with this flow. Upon the completion of a flow's service opportunity, its deficit counter is decremented by the total size of its packets scheduled in the round. It has been shown in [10] that, if the quantum of each flow is proportional to its weight, the relative fairness measure as defined in [9] can be bounded.

The PPLS algorithm approximates the ideal FPLS in a very similar fashion as DRR achieves an approximation of GPS. The PPLS scheduler maintains a linked list of the backlogged flows, *FlowList*. When the scheduler is initialized, *FlowList* is set to an empty list (line 2). For each flow, two variables, instead of one as in DRR, are maintained in the PPLS algorithm: a *processor deficit counter (PDC)* and a *link deficit counter (LDC)*. The link deficit counter is exactly the same as the deficit counter in DRR, which represents the deviation of the bandwidth received by the flow from its ideally fair share. The processor deficit

counter, on the other hand, represents the deviation of the processing time allocated to the flow from its ideally fair share. Thus, each flow in PPLS is assigned two quantum values, a *processor quantum* (PQ) and a *link quantum* (LQ).

When a new packet arrives, the *Enqueue* procedure is invoked (lines 3–10). If this packet comes from a new flow, the *Enqueue* procedure appends this flow to the end of the *FlowList* (line 7) and initializes both of its deficit counters to 0 (lines 8–9).

The *Dequeue* procedure (lines 11–38) functions as follows. It serves all flows in the *FlowList* in a round-robin fashion. When the scheduler visits flow i , it first increments each of the two deficit counters of this flow by the value of the corresponding quantum (lines 16–17). It then verifies whether or not these two deficit counters exceed their upper bounds respectively, and if they do, it resets them to the maximum possible values (lines 18–23). The rationale behind this bounding process will be discussed later in detail. After the deficit counters of flow i are updated, a sequence of packets from flow i are scheduled as long as the total length of these packets is smaller than the link deficit counter, and the total processing cost is smaller than the processing deficit counter, as in the **while** loop in lines 24–33. In the meantime, when a packet is scheduled, both deficit counters are decremented by the corresponding cost of this packet (lines 30–31). Finally, when the scheduler finishes serving a flow and the flow still remains backlogged, the scheduler places the flow back at the end of the *FlowList* (lines 34–36).

Recall that in DRR, for each flow, the quantum is set to be proportional to its weight, therefore, each flow receives in each round, on average, a service of total amount proportional to its weight. In this report, the sum of a certain quantity over *all* flows is denoted by dropping the subscript for the flow in the notation. For example, w is the sum of the weights for all flows, i.e., $w = \sum_i w_i$. Therefore, in DRR, we have

$$\frac{Q_i}{w_i} = \frac{Q}{w}, \forall i.$$

Similarly, in PPLS, the quantum values of each flow are also proportional to its weight, i.e., $\forall i$,

$$\frac{PQ_i}{w_i} = \frac{PQ}{w} \quad (2)$$

$$\frac{LQ_i}{w_i} = \frac{LQ}{w}. \quad (3)$$

Thus the amount of the shared resources each flow is entitled to utilize in each round is guaranteed to be, on average, proportional to its weight. In addition, the ratio of the sum of processing quanta for all flows, PQ , to the sum of link quanta for all flows, LQ , should also be equal to the ratio of the total amount of processing resource to the total amount of link resource in each round, i.e.,

$$\frac{PQ}{LQ} = \frac{R_P}{R_L}. \quad (4)$$

From (2), (3) and (4), it is apparent that,

$$\frac{PQ_i}{LQ_i} = \frac{R_P}{R_L}. \quad (5)$$

In other words, for each flow, the quantum value corresponding to a resource is proportional to the total amount of that resource.

Note that in PPLS, it is possible that the prime resource for flow i remains the same for a long period, and therefore, without the bounding procedure in lines 18–23, the deficit counter for the non-prime resource would reach a large value. For example, consider a case in which the prime resource for flow i has been the processing resource P for a long time and, as a result, the link deficit counter LDC_i is very large. Assume that at this point, the prime resource for flow i switches to the link resource L and, in addition, flow i now consumes almost no processing resource. In such a situation, flow i will be able to have a long sequence of packets scheduled because of its large link deficit counter LDC_i . This would significantly degrade the short-term fairness of the PPLS scheduler. For this reason, we choose to set a maximum threshold on the deficit counter for each resource, in case any specific resource has not been fully utilized for a long time. In cases where short-term fairness is not important, these thresholds may simply be set to infinity. A similar rationale may also be found in the context of fair scheduling in wireless networks where a maximum lag is applied when a flow has not fully utilized its share of the bandwidth [18].

It can be readily verified that if the processor resource P is sufficient for all flows, i.e., the processor resource P never becomes the prime resource for any flow, the PPLS strategy regresses into the DRR policy. It can also be readily verified that, like DRR, the per-packet computing complexity of the PPLS algorithm is $O(1)$, under the condition that for each flow i , $LQ_i > M_L$ and $PQ_i > M_P$ where M_L and M_P are the maximum packet size and the maximum packet processing cost, respectively. The proof of this work complexity is similar to that for DRR [10].

D. Fairness Analysis of PPLS

Our fairness analysis of PPLS is an extension of that in [10]. Recall that in bandwidth sharing, the *fairness measure*, $FM(t_1, t_2)$, is defined as the maximum value of $Sent_i(t_1, t_2)/w_i - Sent_j(t_1, t_2)/w_j$ amongst all pairs of flows (i, j) backlogged in the interval $[t_1, t_2]$, where $Sent_i(t_1, t_2)$ is the total length of the packets transmitted by the scheduler from flow i during interval $[t_1, t_2]$, i.e., the cumulative amount of bandwidth resource allocated to flow i during this interval. In addition, the fairness bound FB is defined as the maximum value of $FM(t_1, t_2)$ for all possible intervals $[t_1, t_2]$ [10]. Note that the dimension of $Sent_i(t_1, t_2)$ is in units of packet length, i.e., bytes. To extend this concept to our situation of multiple heterogeneous resources, we need to normalize it by the peak resource consumption rate.

The *normalized cumulative processor allocation* of flow i during time interval $[t_1, t_2)$, denoted by $CPA_i^n(t_1, t_2)$, is defined as the total amount of the processing resource allocated to flow i during interval $[t_1, t_2)$, normalized by the peak processing rate R_P . The *normalized cumulative link allocation* of flow i during time interval $[t_1, t_2)$, denoted by $CLA_i^n(t_1, t_2)$, is similarly defined, except that the resource associated is the bandwidth.

Note that both the normalized cumulative link allocation and the normalized cumulative processor allocation are in units of time. Therefore, we are able to proceed to define the *normalized cumulative resource allocation* of flow i during time interval $[t_1, t_2)$, denoted by $\text{CRA}_i^n(t_1, t_2)$, as the larger value between the normalized cumulative processor and link allocations of flow i during $[t_1, t_2)$. In other words,

$$\text{CRA}_i^n(t_1, t_2) = \max\{\text{CPA}_i^n(t_1, t_2), \text{CLA}_i^n(t_1, t_2)\}.$$

Now we can extend the definition of the fairness measure as follows:

Definition 8: The *normalized fairness measure* over time interval $[t_1, t_2)$, $\text{FM}^n(t_1, t_2)$, is defined as the maximum value, amongst all pairs of flows (i, j) that are backlogged during time interval $[t_1, t_2)$, of the normalized cumulative resource allocation $\text{CRA}_i^n(t_1, t_2)$. That is,

$$\text{FM}^n(t_1, t_2) = \max_{\forall(i,j)} \left| \frac{\text{CRA}_i^n(t_1, t_2)}{w_i} - \frac{\text{CRA}_j^n(t_1, t_2)}{w_j} \right|.$$

The *normalized fairness bound* FB^n is defined as the maximum value of the normalized fairness measure $\text{FM}^n(t_1, t_2)$ over all possible intervals $[t_1, t_2)$.

Analogous to the case of a single shared resource, if a scheduling algorithm for the joint allocation of processing and bandwidth resources leads to a finite normalized fairness bound, one can conclude that this algorithm approximates the ideally fair allocation and achieves long-term fairness. The following theorem states this about the PPLS algorithm.

Theorem 2: The normalized fairness bound of PPLS is a finite constant.

Proof: The reader is referred to the Appendix II. ■

V. SIMULATION RESULTS AND ANALYSIS

Our simulation model consists of 8 flows with equal weights sharing a processor P and a link L , as shown in Fig. 1(a). Five different scheduling policies including the PPLS algorithm are implemented.

- FCFS (First-Come First-Served): A simple FCFS scheme is used. The scheduling order is only determined by the packet timestamps.
- PPLS: When the PPLS algorithm is implemented, a FCFS strategy is used on the buffer between the processor P and the link L , since the order of the packets has already been determined by the PPLS algorithm.
- LDRR (Link Deficit Round Robin): A DRR algorithm in the allocation of only the link bandwidth is implemented (i.e., the original DRR).
- PDRR (Processor Deficit Round Robin): A DRR algorithm in the allocation of only the processing resource is implemented.
- DDRR (Double Deficit Round Robin): Two DRR schedulers are used. PDRR is used before the processor P and LDRR is used before the link L . Note that this is the only scheme in which a scheduler is implemented between the processor and the link.

TABLE II

THE RATIO OF THE PROCESSING RESOURCE TO THE LINK RESOURCE REQUIRED BY EACH FLOW IN OUR STUDY USING SYNTHETIC TRAFFIC.

Flow ID	P/L Ratio (in cycles/byte)
1	1
2	2
3	3
4	4
5	1
6	1/2
7	1/3
8	1/4

In the following, we describe two sets of simulation experiments. In the first set of experiments, a synthetic traffic sequence is used, while the second set uses real gateway traffic traces.

A. Synthetic Traffic

In our first study, we use synthetic traffic to test the fairness properties of the PPLS algorithm under some extreme situations. For each flow, the ratio of the amount of the processing resource required to the amount of the bandwidth resource required is a fixed value. Note that in the definition of the normalized fairness measure, $\text{FM}^n(t_1, t_2)$, if both the R_P and R_L are multiplied by the same value, the normalized fairness measure will also be multiplied by this value. In other words, the fact of whether or not the normalized fairness measure is bounded does not change except that the bound itself may vary. Therefore, for better illustration and easier comparison, we normalize the resource amounts such that the average number of processor cycles needed per packet is numerically equal to the average number of bytes per packet. Table II shows the ratios used in this study. Note that flows 1 and 5 have equal normalized demands for both resources. The prime resource for flows 2, 3 and 4 is the processor, and that for flows 6, 7 and 8 is the link. For flows 1 to 4, the sizes of packets generated is uniformly distributed between 1 and 1,600 bytes, while for flows 5 to 8, the processing cost is uniformly distributed between 1 and 1,600 cycles. Therefore, the maximum packet size is 6,400 bytes and the maximum processing cost is 6,400 cycles. These maximum values are the quantum values assigned to each flow.

Fig. 4 shows the normalized cumulative resource allocations, $\text{CRA}_i^n(0, \tau)$, for all flows i after a long run in the simulations. In this plot, the fairer an allocation policy, the closer its corresponding curve to a straight horizontal line. It is apparent from this figure that the PPLS scheduling policy does achieve good fairness. Note that, as expected, the FCFS scheme is the worst among all in terms of fairness. Regarding LDRR and PDRR, each can achieve fair distribution of the normalized cumulative allocation with respect to a certain resource, but not the overall normalized cumulative resource allocation. Take LDRR as an example. It achieves fair distribution of the normalized cumulative link allocation for all flows. Therefore, flows with the processor as the prime resource, namely flows 2 to 4 in this case,

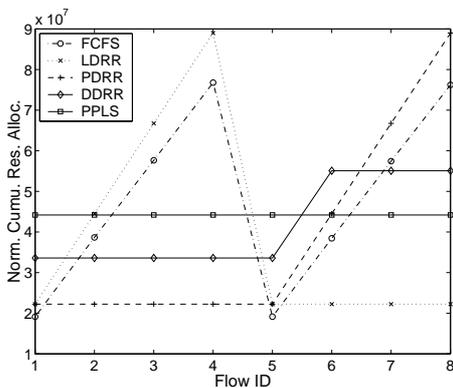


Fig. 4. The simulation results using synthetic traffic. In this plot, the fairer an allocation policy, the closer its corresponding curve to a straight horizontal line.

receive a larger amount of normalized cumulative processor allocation as compared to other flows. This indicates that LDRR fails to achieve overall fairness among flows. PDRR functions exactly in the opposite way: it fairly distributes the normalized cumulative processor allocation among all flows, but flows with the link as the prime resource (flows 5 to 8) receive a large normalized cumulative link allocation. Thus, PDRR also fails to achieve overall fairness.

One interesting observation is regarding the DRRR scheme. Intuitively, one may expect DRRR to serve as a fair scheduler in the allocation of processing and bandwidth resources, since it has two schedulers, one fair with respect to the processor and the other fair with respect to the link. However, Fig. 4 shows that this is not the case. This is because the DRRR scheme implements the two fair schedulers in different stages. Note that the PDRR scheduler before the processor P is responsible for fairly allocating the normalized cumulative processor allocation to all flows. That means, at this point, more packets (in bytes) from those flows with the link as the prime resource (flows 6 to 8) will be scheduled from the processor P . On the other hand, those flows with the processor as the prime resource (flows 2 to 4) will not have enough packets remaining backlogged in the buffer before the link L . The LDRR scheduler, therefore, finds flows 2 to 4 to be relatively idle and ends up transmitting more packets from flows 6 to 8, thus causing a higher normalized cumulative resource allocation for flows 6 to 8. In fact, the DRRR scheme allocates resources fairly to all flows with the same prime resource, but favors the flows with the “final” resource as the prime resource.

B. Gateway Traffic Traces

In this study, we use real traffic recorded at an Internet gateway as the input traffic [14, 19].² The traffic traces include the processing delay (in milliseconds) for each packet, along with

²Global Positioning System technology was used to precisely record the timestamp of each packet at each node. In the trace data, filtered IP headers were examined to track the same packet at different nodes. The difference between the timestamps of the same packet at adjacent nodes was computed as the delay. The link speed connecting these nodes was taken into consideration so that the transmission delay of each packet was removed from the recorded delay. Note that this delay was still the sum of the processing delay and the queuing delay. However, it was noticed that for traffic in one of the two direc-

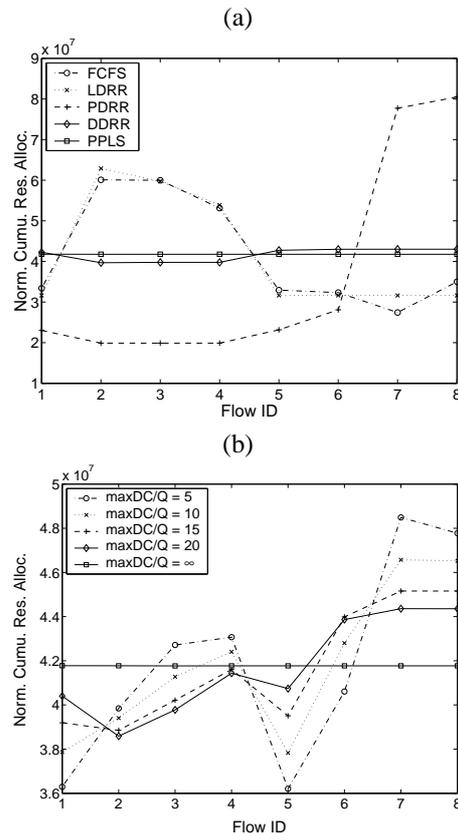


Fig. 5. The simulation results using gateway traffic traces. (a) The case when $\max PDC_i$ and $\max LDC_i$ are large enough for all i such that lines 19 and 22 in Fig. 3 are never executed. (b) The effect of $\max PDC_i$ and $\max LDC_i$ in the PPLS algorithm. Here, for all i , we assume $\max PDC_i = \max LDC_i = \max DC$ and $PQ_i = LQ_i = Q$. Again, in these plots, the fairer an allocation policy, the closer its corresponding curve to a straight horizontal line.

the packet size (in bytes). For our experiments, we assume a fixed processing rate, and correspondingly convert the processing delay of each packet into processor cycles. Again, we convert the processing delay of each packet in such a way that the average processing delay per packet (in units of cycles) is numerically equal to the average size per packet (in units of bytes). To achieve a better comparison with the previous study using synthetic traffic, the flows have been ordered in such way that the overall prime resource for flows 1 to 4 is the processor, and that for flows 5 to 8 is the link.

Fig. 5(a) illustrates the normalized cumulative resource allocation for the five scheduling policies in this experiment. Again, the PPLS algorithm performs very well in terms of fairness. It is observed that all other conclusions drawn from the synthetic study are still valid.

Note that in this study, the DRRR scheme performs closer to the PPLS algorithm than in the previous study. This can be attributed to the fact that in real traces, the demands of each flow for the processing and bandwidth resources are more balanced than those in the synthetic traffic. However, this would not be valid if a flow has a dominant demand for one resource in comparison to another, as might happen during a DoS attack. In

tions, the queue occupancy was never above 1 packet, and this eliminates the queuing delay and validates the use of this delay as the pure processing delay.

addition, the PPLS algorithm only needs one scheduler in real implementation while the DRR needs two.

C. Effect of Maximum Deficit Counter Values

Note that in the synthetic study, no flow changes its prime resource during the experiment. Therefore, the setting of the maximum values of the deficit counters ($maxPDC_i$ and $maxLDC_i$) in the PPLS algorithm has no effect on the outcome of the simulations. Next, using the real gateway traces, we focus on the effect of setting the maximum values of deficit counters in the PPLS algorithm. This is shown in Fig. 5(b), where we assume that, for all i , $maxPDC_i$ (in units of processing cycles) is numerically equal to $maxLDC_i$ (in units of bytes).

It is apparent that the prime resource of a flow changes in this study, since the normalized cumulative resource allocation begins to show differences under the PPLS algorithm. However, it should be noticed that the normalized cumulative resource allocations for the flows are still reasonably close to each other, due to the long-term fairness achieved by the PPLS algorithm.

From Fig. 5(b), it is observed that, as expected, the long-term fairness among normalized cumulative resource allocations degrades as the set maximum values of the deficit counters decrease. For example, when the maximum deficit counters are set to be 10 times as large as the quantum values, the normalized cumulative resource allocation exhibits a 10% variation from the ideal.

It is also observed from Fig. 5(b) that flows with more balanced normalized cumulative allocations between the two resources over the long run, such as flows 1, 2 and 5, are likely to receive less normalized cumulative resource allocation. This may be attributed to the fact that these flows are more likely to temporarily change the prime resource, and therefore, setting the deficit counter for the current non-prime resource to the maximum value may reduce the future usage of this resource when it later becomes the prime resource. On the other hand, the unbalanced flows are less likely to temporarily change the prime resource, and therefore, the effect on these flows of setting the deficit counter for the non-prime resource to the maximum value is limited. Similar scenarios may also be found in other situations, such as bandwidth sharing. For example, in DRR, a flow that frequently changes its status (between being backlogged and not being backlogged) will be sacrificed in the long run, since each time it becomes non-backlogged its unused deficit counter is reset to 0, thus causing it to lose bandwidth share.

Based on the above discussion, by setting appropriate maximum values for the deficit counters, one can tune the trade-offs between long-term and short-term fairness achieved by the PPLS algorithm. This is similar to the role played by the maximum lag in wireless scheduling [18].

VI. CONCLUDING REMARKS

A. Summary

Research in fair allocation of the bandwidth resource has been active for decades. Traffic flows, however, encounter multiple resources other than bandwidth, including processor, buffer, and power, as they traverse the network. Further, the

bandwidth resource is not always the sole bottleneck causing network congestion. In this report, we consider a set of shared resources which are *essential and related* such as processor, link bandwidth, and power. We then present the *Principle of Fair Essential Resource Allocation*, or the FERA principle, which defines the fairness in the joint allocation of these resources. We further apply the FERA principle into a system consisting of a shared processor and a shared link, and propose a practical and *provably* fair algorithm, the *Packet-by-packet Processor and Link Sharing (PPLS)*, for the joint allocation of the processor and bandwidth resources. It is our hope that this work will facilitate future research on achieving *provable* fairness in computer networks.

B. A Discussion on Implementation of PPLS

In this report, we select DRR [10] as the starting point in the design of the fair allocation policy for a shared processor and a shared link, because of its relatively simple implementation. Other fair scheduling algorithms can be also used, such as Weighted Fair Queueing (WFQ) [3], Worst-case Fair Weighted Fair Queueing (WF²Q) [11], Surplus Round Robin (SRR) [20], and Elastic Round Robin (ERR) [12].

Note that in many situations, the processing cost of a packet cannot be determined before it is actually processed. If this is the case, one can have the following choices to modify the PPLS algorithm. The first way is to let the scheduler predict the processing cost, and make scheduling decisions based on predicted values. In the second choice, the scheduler serves the packet first, then updates the deficit counters accordingly. In this way, it is possible that after serving a packet, its processing deficit counter becomes negative, thus breaking the fairness property of the PPLS algorithm. Therefore, the scheduler needs an additional counter to record the minimum normalized deficit counter for all flows, and if this value becomes negative, at the beginning of next round, it needs to add a proper amount to the deficit counter of each flow to make it non-negative. Note that even when using prediction before scheduling, one still needs this protection from a negative deficit counter. Therefore, one can combine these two approaches: predict first, and then correct if not accurate. In using this method, the PPLS algorithm would incorporate some of the principles of the ERR scheduler [12], where the resource requirements are not assumed to be known prior to the allocation.

C. Discussions on Further Extensions

Note that in our study, it is assumed that each flow has a unique weight which determines its relative rightful share for each resource. If instead, for each flow, a different weight is associated with each individual resource, the premise of this work can still be applied. The only difference would be that when defining the prime resource for each flow, the weight for each individual resource needs to be taken into consideration. This requires an additional concept, the *prime weight*, defined as the weight associated with the prime resource. In this case, for all flows, the quantum values for each resource in the PPLS algorithm need to be proportional to the weights corresponding to that resource.

Even though we have focused only on the processing and bandwidth resources, the FERA principle and the design of the PLS algorithm may be easily extended to systems with additional essential and related resources, such as a wireless system where processor, link, and power are all shared. In this case, for each flow, three quanta and three deficit counters are needed. The core of the algorithm, however, remains the same, i.e., a packet from a flow can be scheduled only if all three deficit counters of the flow are large enough.

REFERENCES

- [1] D. Cohen and K. Narayanaswamy, "A fair service approach to defending against packet flooding attacks," <http://www.cs3-inc.com/ddos.html>.
- [2] D. P. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1991.
- [3] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. ACM SIGCOMM*, Austin, TX, Sep. 1989, pp. 1–12.
- [4] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks – the single node case," in *Proc. IEEE INFOCOM*, Florence, Italy, May 1992, pp. 915–924.
- [5] S. Keshav, *An Engineering Approach to Computer Networking: ATM Network, the Internet, and the Telephone Network*, Addison-Wesley, Reading, MA, 1997.
- [6] F. Kelly, "Charging and rate control for elastic traffic," *Europ. Trans. Telecom.*, vol. 8, no. 1, pp. 33–37, Jan. 1997.
- [7] Z. Cao and E. W. Zegura, "Utility max-min: An application-oriented bandwidth allocation scheme," in *Proc. IEEE INFOCOM*, New York, NY, Mar. 1999, pp. 793–801.
- [8] L. Kleinrock, *Queueing Systems*, vol. 2, Computer Applications, Wiley, New York, NY, 1976.
- [9] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, Toronto, Canada, Jun. 1994, pp. 636–646.
- [10] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 375–385, Jun. 1996.
- [11] J. C. R. Bennett and H. Zhang, "WF²Q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 1996, pp. 120–128.
- [12] S. S. Kanhere, H. Sethu, and A. B. Parekh, "Fair and efficient packet scheduling using elastic round robin," *IEEE Trans. Parall. Distr. Syst.*, vol. 13, no. 3, pp. 324–336, Mar. 2002.
- [13] J. M. Blanquer and B. Özden, "Fair queueing for aggregated multiple links," in *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001, pp. 189–197.
- [14] K. Mochalski, J. Micheel, and S. Donnelly, "Packet delay and loss at the Auckland Internet access path," in *Proc. Passive Active Measure. Workshop*, Fort Collins, CO, Mar. 2002.
- [15] V. Raghunathan, S. Ganeriwal, C. Schurgers, and M. Srivastava, "E²WFQ: An energy efficient fair scheduling policy for wireless systems," in *Proc. Int. Symp. Low Power Electr. Design*, Monterey, CA, Aug. 2002, pp. 30–35.
- [16] Y. Zhou and H. Sethu, "Toward end-to-end fairness: A framework for the allocation of multiple prioritized resources," in *Proc. IEEE Int. Perf. Comput. Commun. Conf.*, Phoenix, AZ, Apr. 2003, pp. 495–504.
- [17] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Services*, Dec. 1998, IETF RFC 2475, <http://www.ietf.org/rfc/rfc2475.txt>.
- [18] S. Lu, V. Bhargavan, and R. Srikant, "Fair scheduling in wireless packet networks," *IEEE/ACM Trans. Networking*, vol. 7, no. 4, pp. 473–489, Aug. 1999.
- [19] WAND Research Group, "Auckland-VI trace data," <http://pma.nlanr.net/Traces/long>.
- [20] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Networking*, vol. 3, no. 4, pp. 365–386, Aug. 1995.
- [21] S. Lu, V. Bhargavan, and R. Srikant, "Fair scheduling in wireless packet networks," in *Proc. ACM SIGCOMM*, Cannes, France, Sep. 1997, pp. 63–74.

APPENDIX I PROOF OF THEOREM 1

Lemma 1: In a system with multiple essential and related resources, the normalized allocations received by a flow i are identical under two allocation policies q and s if $\mathcal{A}_i^q = \mathcal{A}_i^s$.

Proof: Let \mathcal{B}_i^q and \mathcal{B}_i^s be one of the prime resources of flow i under policies q and s , respectively. We have,

$$\begin{aligned} A_{i, \mathcal{B}_i^s}^q &\leq A_{i, \mathcal{B}_i^q}^q \\ A_{i, \mathcal{B}_i^q}^s &\leq A_{i, \mathcal{B}_i^s}^s. \end{aligned}$$

In addition, we have $A_{i, \mathcal{B}_i^q}^q = A_{i, \mathcal{B}_i^s}^s$ since $\mathcal{A}_i^q = \mathcal{A}_i^s$. Therefore $A_{i, \mathcal{B}_i^s}^q \leq A_{i, \mathcal{B}_i^s}^s$ which means flow i receives less allocation of resource \mathcal{B}_i^s under policy q than under policy s . Also, since the resources are essential, flow i receives no more allocation of any other resource including \mathcal{B}_i^q under policy q than under policy s , i.e.,

$$A_{i, \mathcal{B}_i^q}^s \geq A_{i, \mathcal{B}_i^q}^q = A_{i, \mathcal{B}_i^s}^s = A_{i, \mathcal{B}_i^s}^q.$$

This means \mathcal{B}_i^q is also one of the prime resources of flow i under policy s . It may be similarly deduced that \mathcal{B}_i^s is one of the prime resources of flow i under policy q . In summary, if $\mathcal{A}_i^q = \mathcal{A}_i^s$, the sets of prime resources of flow i are identical under policies q and s if $\mathcal{A}_i^q = \mathcal{A}_i^s$, and the allocations of these prime resources to flow i are also identical under policies q and s . Since the resources are related, flow i receives identical allocations of all resources from both policies q and s . ■

Now, we will proceed to prove Theorem 1 by contradiction.

Assume that in the considered system, when applying the FERA principle, two different policies q and s are both fair work-conserving allocation policies corresponding to the notion of fairness \mathcal{F} . Denote the constraints corresponding to these two allocation policies by C^q and C^s , respectively.

Note that for these two allocation policies q and s , the vectors of the largest normalized allocations, i.e., $[\mathcal{A}_i^q]$ and $[\mathcal{A}_i^s]$, cannot be equal. This is because if that is the case, from Lemma 1, the allocated amount of each resource for each flow will be the same under policies q and s , and policies q and s will be identical.

Since the system under consideration remains the same, we know that both the vector of the largest normalized demands $[\mathcal{D}_i]$ and the vector of the flow weights $[w_i]$ are the same. From the definition of a uniquely deterministic notion of fairness, we know that $C^q \neq C^s$ since, otherwise, the two vectors of largest normalized allocations, $[\mathcal{A}_i^q]$ and $[\mathcal{A}_i^s]$, will be equal.

Since the notion of fairness \mathcal{F} is non-decreasing, from the definition, we have either $[\mathcal{A}_i^q] \prec [\mathcal{A}_i^s]$ or $[\mathcal{A}_i^s] \prec [\mathcal{A}_i^q]$. Without loss of generality, we assume that $[\mathcal{A}_i^q] \prec [\mathcal{A}_i^s]$, i.e.,

$$A_i^q \leq A_i^s, \forall i. \quad (6)$$

Therefore, for all flows, the allocated amount of each resource under policy q is no more than that under policy s , since the resources are related and essential.

In addition, there must exist at least one flow, for which (6) is not an equality. In other words, there exists at least one flow, which gets more resources under policy s than under policy q .

Hence, under policy s , as opposed to policy q , no flow gets less allocation for any resource, and at least one flow is allocated more of some resources. This violates the assumption that policy q is work-conserving and completes the proof.

APPENDIX II PROOF OF THEOREM 2

Without loss of generality, we assume that the flow weights are normalized in such a way that the smallest of the weights assigned to a flow is 1.

In the rest of this proof, we will limit our consideration to the situations where lines 19 and 22 in Fig. 3 are never executed, i.e., the deficit counters of any flow are never above the thresholds. The reason of this assumption is similar to the one used in the design of Idealized Wireless Fair Queueing (IWFQ), where fairness in bandwidth cannot be guaranteed if any flow lags more than the maximum lag allowed by the wireless packet scheduler [21].

Lemma 2: In an execution of the PPLS strategy, at the end of each round k , for any flow i ,

- 1) The following two statements are always satisfied:

$$0 \leq PDC_i(k) \leq \max_{\forall i} \{maxPDC_i\}$$

$$0 \leq LDC_i(k) \leq \max_{\forall i} \{maxLDC_i\};$$

- 2) At least one of the following statements is always satisfied:

$$0 \leq PDC_i(k) \leq M_P$$

$$0 \leq LDC_i(k) \leq M_L$$

where M_P and M_L are, respectively, the maximum processing cost of a packet and the maximum link cost of a packet.

Proof: First, it can be readily verified that the deficit counters can never be negative. The first half of Lemma 2 can be directly derived from the assumption that lines 19 and 22 are never executed.

Next we prove the second half of Lemma 2 by contradiction. Assume that both statements are not true, then we have $PDC_i(k) > M_P$ and $LDC_i(k) > M_L$. Note that at this moment, flow i still has packets in the queue waiting to be scheduled. Otherwise, both deficit counters of flow i should be reset to 0. Consider the head-of-line packet of flow i , say p . Apparently its processing cost is no more than M_P and its link cost is no more than M_L . In other words, its processing cost is less than $PDC_i(k)$ and its link cost is less than $LDC_i(k)$, and therefore, based on the PPLS algorithm, packet p should be scheduled in round k . This violates the assumption that packet p is the head-of-line packet from flow i at the end of round k . This completes the proof. ■

Lemma 2 readily leads to the following Corollary.

Corollary 1: In an execution of the PPLS strategy, at the end of each round k , for any flow i ,

$$\max \left\{ \frac{PDC_i(k)}{R_P}, \frac{LDC_i(k)}{R_L} \right\} \leq \alpha$$

$$\min \left\{ \frac{PDC_i(k)}{R_P}, \frac{LDC_i(k)}{R_L} \right\} \leq \beta$$

where constants α and β are defined as follows:

$$\alpha = \max \left\{ \frac{\max_{\forall i} \{maxPDC_i\}}{R_P}, \frac{\max_{\forall i} \{maxLDC_i\}}{R_L} \right\} \quad (7)$$

$$\beta = \min \left\{ \frac{M_P}{R_P}, \frac{M_L}{R_L} \right\}. \quad (8)$$

According to (5), we also define constant γ as follows,

$$\gamma = \frac{\min_{\forall i} \{PQ_i\}}{R_P} = \frac{\min_{\forall i} \{LQ_i\}}{R_L}. \quad (9)$$

Lemma 3: During an execution of the PPLS strategy over any m rounds, for any flow i ,

$$mw_i\gamma - \beta \leq CRA_i^n(m) \leq mw_i\gamma + \alpha$$

where α, β, γ are constants defined in (7), (8) and (9), respectively.

Proof: Denote by $SCPA_i(k)$ the cumulative processor allocation of flow i in a single round k . From the algorithm we have,

$$SCPA_i(k) = PQ_i + PDC_i(k-1) - PDC_i(k).$$

This leads to

$$\begin{aligned} CPA_i(m) &= \sum_{k=1}^m SCPA_i(k) \\ &= mPQ_i + PDC_i(0) - PDC_i(m) \end{aligned}$$

and

$$\begin{aligned} CPA_i^n(m) &= \frac{CPA_i(m)}{R_P} \\ &= m \frac{PQ_i}{R_P} + \frac{PDC_i(0) - PDC_i(m)}{R_P}. \end{aligned}$$

From (2) we have,

$$\frac{PQ_i}{R_P} = \frac{w_i}{\min_{\forall j} \{w_j\}} \frac{\min_{\forall j} \{PQ_j\}}{R_P} = w_i\gamma$$

and therefore

$$CPA_i^n(m) = mw_i\gamma + \frac{PDC_i(0) - PDC_i(m)}{R_P}.$$

Since both $PDC_i(0)$ and $PDC_i(m)$ are non-negative,

$$mw_i\gamma - \frac{PDC_i(m)}{R_P} \leq CPA_i^n(m) \leq mw_i\gamma + \frac{PDC_i(0)}{R_P}.$$

Similarly we have

$$mw_i\gamma - \frac{LDC_i(m)}{R_L} \leq CLA_i^n(m) \leq mw_i\gamma + \frac{LDC_i(0)}{R_L}.$$

Applying the above in the definition of normalized cumulative resource allocation leads to the following:

$$\begin{aligned} \text{CRA}_i^n(m) &\leq mw_i\gamma + \max\left\{\frac{PDC_i(0)}{R_P}, \frac{LDC_i(0)}{R_L}\right\} \\ \text{CRA}_i^n(m) &\geq mw_i\gamma - \min\left\{\frac{PDC_i(m)}{R_P}, \frac{LDC_i(m)}{R_L}\right\}. \end{aligned}$$

Applying Corollary 1 into the above inequalities completes the proof. ■

Consider a certain time interval $[t_1, t_2)$, during which all flows remain backlogged. Consider any pair of flows i and j . Assume that during $[t_1, t_2)$, flow i receives m_i rounds of service while flow j receives m_j rounds of service. Since both flows i and j are backlogged during time interval (t_1, t_2) , and the scheduler serves the flows in a round-robin fashion, we have $|m_i - m_j| \leq 1$.

Applying Lemma 3 we have

$$\begin{aligned} \frac{\text{CRA}_i^n(t_1, t_2)}{w_i} &\leq m_i\gamma + \frac{\alpha}{w_i} \\ \frac{\text{CRA}_j^n(t_1, t_2)}{w_j} &\geq m_j\gamma - \frac{\beta}{w_j} \end{aligned}$$

Therefore,

$$\begin{aligned} &\frac{\text{CRA}_i^n(t_1, t_2)}{w_i} - \frac{\text{CRA}_j^n(t_1, t_2)}{w_j} \\ &\leq (m_i - m_j)\gamma + \frac{\alpha}{w_i} + \frac{\beta}{w_j} \\ &\leq \alpha + \beta + \gamma. \end{aligned}$$

Similarly we can also derive that,

$$\frac{\text{CRA}_j^n(t_1, t_2)}{w_j} - \frac{\text{CRA}_i^n(t_1, t_2)}{w_i} \leq \alpha + \beta + \gamma.$$

Since flows i and j can be any pair of flows, based on the definition of the normalized fairness measure we have,

$$\text{FM}^n(t_1, t_2) \leq \alpha + \beta + \gamma.$$

Note that α , β and γ are all finite constants. Therefore, $\text{FM}^n(t_1, t_2)$ is bounded by a finite constant over any time interval during which all flows are backlogged, i.e., the fairness bound FB^n exists for the PPLS strategy and it is finite. This proves the statement of Theorem 2.