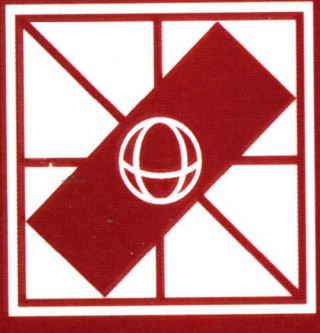
Georg Carle
Martina Zitterbart (Eds.)

LNCS 2334

Protocols for High Speed Networks

7th IFIP/IEEE International Workshop, PfHSN 2002
Berlin, Germany, April 2002
Proceedings



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Georg Carle
Fraunhofer Institut FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
E-mail: carle@fokus.gmd.de

Martina Zitterbart
University of Karlsruhe
Faculty of Computer Science, Institute of Telematics
Zirkel 2, 76128 Karlsruhe, Germany
E-mail: zit@tm.uka.de

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Protocols for high speed networks : 7th IFIP/IEEE international workshop ; proceedings / PfHSN 2002, Berlin, Germany, April 22 - 24, 2002. Georg Carle ; Martina Zitterbart (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2334)
ISBN 3-540-43658-8

CR Subject Classification (1998): C.2, D.4.4, H.3.5, K.4.4

ISSN 0302-9743
ISBN 3-540-43658-8 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN 10869765 06/3142 5 4 3 2 1 0

Preface

This workshop on "Protocols for High-Speed Networks" is the seventh in a successful series of international workshops, well known for their small and focused target audience, that provide a sound basis for intensive discussions of high-quality and timely research work.

The location of the workshop has alternated between Europe and the United States, at venues not only worth visiting for the workshop, but also for the distinct impressions they leave on the participants. The first workshop was held in 1989 in Zurich. Subsequently the workshop was moved to Palo Alto (1990), Stockholm (1993), Vancouver (1994), Sophia-Antipolis/Nice (1996), and Salem (1999). In 2002, the workshop was hosted in Berlin, the capital of Germany.

PfHSN is a workshop providing an international forum that focuses on issues related to high-speed networking, such as protocols, implementation techniques, router design, network processors and the like. Although the topics have shifted during the last couple of years, for example, from parallel protocol implementations to network processors, it could be observed that high speed remains a very important issue with respect to future networking. Traditionally, PfHSN is a relatively focused and small workshop with an audience of about 60 participants. The workshop is known for lively discussions and very active participation of the attendees. A significant component of the workshop is the institution of so-called Working Sessions chaired by distinguished researchers focusing on topical issues of the day. The Working Sessions, introduced in 1996 by Christophe Diot and Wallid Dabbous, have proved to be very successful, and they contribute considerably to making PfHSN a true "workshop."

This year, the program committee had to be once again rather selective, accepting only 14 out of 54 submissions as full papers. Working sessions on extremely timely issues, e.g., High-Speed Mobile Wireless, complemented the program. In addition, the workshop featured a keynote speech which gave an operator's viewpoint on high-speed networking, and an invited talk bringing a manufacturer's viewpoint. In honor of the large number of good submissions and to allow for the presentation of new and innovative work, the program was complemented by a set of six short papers and a panel session.

High-speed networking has changed enormously during the thirteen years covered by the workshop. Technologies such as ATM have moved into the spotlight and out again. What was once at the forefront of technology and deployed only in niches has become a commodity, with widespread availability of commercial products such as Gigabit Ethernet. At the same time, many issues identified by research to be important a decade ago have proven to be very timely today.

While this year's papers give answers to many important questions, they also show that there is still a lot of room for additional work in the future.

March 2002 Georg Carle, Martina Zitterbart

Organization

The Seventh International Workshop on Protocols for High-Speed Networks (PfHSN 2002), held in Berlin, Germany from Monday, April 22 to Wednesday, April 24, 2002, was jointly organized by the Fraunhofer Institute FOKUS and the Institute of Telematics, University of Karlsruhe. It was sponsored by IFIP WG6.2, the Working Group on Network and Internetwork Architecture. Technical co-sponsorship was provided by the IEEE Communications Society Technical Committee on Gigabit Networking. The workshop was organized in cooperation with COST Action 263 – Quality of future Internet Services.

Workshop Co-chairs

Georg Carle, Fraunhofer FOKUS, Germany
Martina Zitterbart, University of Karlsruhe, Germany

Steering Committee

James P.G. Sterbenz, BBN Technologies, GTE, USA (Chair)
Per Gunningberg, Uppsala University, Sweden
Byran Lyles, Sprint Labs, USA
Harry Rudin, IBM Zurich Research Lab, Switzerland
Martina Zitterbart, University of Karlsruhe, Germany

Program Committee

Sujata Banerjee, HP Labs and Univ. of Pittsburgh, USA
Olivier Bonaventure, University of Namur, Belgium
Torsten Braun, University of Bern, Switzerland
Georg Carle, Fraunhofer FOKUS, Germany
Jon Crowcroft, UCL, UK
Christophe Diot, Sprint Labs, USA
Julio Escobar, Centauri Technologies Corporation, Panama
Serge Fdida, University P. and M. Curie, Paris, France
Per Gunningberg, Uppsala University, Sweden
Marjory Johnson, RIACS/NASA Ames Research Center, USA
Guy Leduc, Univ. of Liege, Belgium
Jörg Liebeherr, University of Virginia, USA
Byran Lyles, Sprint Labs, USA
Gerald Neufeld, Redback Networks, USA
Luigi Rizzo, University of Pisa, Italy
Harry Rudin, IBM Zurich Research Lab, Switzerland

Patricia Sagmeister, IBM Zurich Research Lab, Switzerland
Jochen Schiller, FU Berlin, Germany
James P.G. Sterbenz, BBN Technologies, GTE, USA
Burkhard Stiller, ETH Zurich, Switzerland
Heinrich Stüttgen, NEC Labs, Heidelberg, Germany
Joe Touch, USC/ISI, USA
Giorgio Ventre, University of Napoli, Italy
Martina Zitterbart, University of Karlsruhe, Germany

Additional Reviewers

Maurizio D'Arienzo, University of Napoli, Italy
Christopher Edwards, Lancaster University, UK
Marcello Esposito, University of Napoli, Italy
Jan Gerke, ETH Zurich, Switzerland
Ilias Iliadis, IBM Zurich Research Lab, Switzerland
H. Hasan, ETH Zurich, Switzerland
David Hausheer ETH Zurich, Switzerland
Rajesh Krishnan, BBN Technologies, USA
Sung-Ju Lee, HP Laboratories Palo Alto, USA
Jan Van Lunteren, IBM Zurich Research Lab, Switzerland
Cristel Pelsser, University of Namur, Belgium
Roman Pletka, IBM Research, Switzerland
Pierre Reinbold, University of Namur, Belgium
Simon Pietro Romano, University of Napoli, Italy
Sambit Sahu, IBM Research, Switzerland
Kave Salamatian, LIP6, University of Paris, France
Steve Uhlig, University of Namur, Belgium

Sponsoring Institutions

T-Systems Nova Berkom, Berlin, Germany
Siemens AG, Information and Communication Networks, Munich, Germany
Network Laboratories Heidelberg, NEC Europe Ltd., Heidelberg, Germany

Table of Contents

Signalling and Controlling

Application-Level Mechanisms

TCP and High Speed Networks

Quality of Service

Traffic Engineering and Mobility

Working Sessions

Invited Paper

A Core-Stateless Utility Based Rate Allocation Framework

Narayanan Venkitaraman, Jayanth P. Mysore, and Mike Needham

Networks and Infrastructure Research, Motorola Labs,
{venkitar,jayanth,needham}@labs.mot.com

Abstract. In this paper, we present a core-stateless framework for allocating bandwidth to flows based on their requirements which are expressed using utility functions. The framework inherently supports flows with adaptive resource requirements and intra-flow drop priorities. The edge routers implement a labeling algorithm which in effect embeds partial information from a flow's utility function in each packet. The core routers maintain no per-flow state. Forwarding decisions are based a packets label and on a threshold utility value that is dynamically computed. Thus the edge and core routers work in tandem to provide bandwidth allocations based on a flow's utility function. We show how the labeling algorithm can be tailored to provide different services like weighted fair rate allocations. We then show the performance of our approach using simulations.

1 Introduction

The Internet is being increasingly used for carrying multimedia streams that are sensitive to the end-to-end rate, delay and drop assurances they receive from the network. We are motivated by two key characteristics that a significant number of these flow share. First, multimedia flows are increasingly becoming adaptive and can adjust their level of performance based on the amount of resource available. The different levels of performance result in varying levels of satisfaction for the user. Another key characteristic is that most of them tend to be composed of packets which contribute varying amounts of utility to the flow they belong to. This intra-flow heterogeneity in packet utility could be caused due to the stream employing a hierarchical coding mechanism as in MPEG or layered multicast, or due to other reasons such as the specifics of a rate adaptation algorithm (as explained later for TCP). In either case, dropping the "wrong" packet(s) can significantly impact the qualitative and quantitative extent to which a flow is able to make use of the resources allocated to it. That being the case, the utility provided by a quantum of resource allocated to a flow depends on the value of the packets that use it. So, merely allocating a certain quantity of bandwidth to a flow does not always imply that the flow will be able to make optimal use of it at all times. As has been observed previously, applications don't care about

G. Carle and M. Zitterbart (Eds.): PfHSN 2002, LNCS 2334, pp. 1–16, 2002.

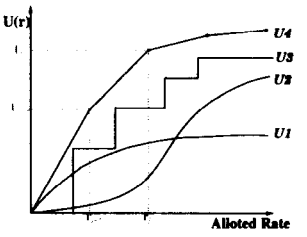


Fig. 1. Utility Functions

bandwidth, per se, except as a means to achieve user satisfaction. In summary, if optimizing the perceived quality of service is the end goal of an architecture, then it is important that we allocate resources according to user's preferences, and provide simple ways for a flow to make best use of its share.

Utility functions have long been recognized as an abstraction for a user to quantify the utility or satisfaction that (s)he derives when a flow is allocated a certain quantum of resource. It maps the range of operational points of a flow to the utility that a user derives at each point. Figure 1 shows some sample utility functions. Such an abstraction provides the necessary flexibility to express arbitrarily defined requirements. Also, it is now well established that different notions of fairness can be defined in terms of utility functions [4, 7, 8, 10]. Partly motivated by recent work by Gibbens, Kelly and others [5, 4, 11, 2], we use utility functions as an abstraction that is used to convey application/user level performance measures to the network. In this paper, we only concern ourselves with allocation of bandwidth as a resource. Therefore, we have used the terms resource and bandwidth interchangeably. We hope that the proposed framework will be a step toward a more general solution that can be used for allocation of other network resources such as those that impact end to end delay and jitter.

In this paper, we propose a scalable framework for allocating bandwidth to flows based on their utility functions. The architecture is characterized by its simplicity – only the edge routers maintain a limited amount of per flow state, and label the packets with some per-flow information. The forwarding behavior at a router is based on the state in the packet header. As the core routers do not perform flow classification and state management they can operate at very high speeds. Furthermore, the framework allows a flow to indicate the relative priority of packets within its stream. The dropping behavior of the system is such that for any flow lower priority packets are dropped preferentially over high priority packets of the flow.

We refer to our architecture as the *Stateless Utility based Resource allocation Framework*(SURF)[1]. The network objective and architecture are described in Section 2. The algorithms implemented by this architecture are described in Section 3. In Section 4, we present the performance of our architecture in a variety of scenarios. Section 5 discusses our implementation experience and some key issues. Section 6 discusses the related work and section 7 concludes the paper.

[1] We borrow the notion of stateless core from CSFQ [14]

2 System Architecture

2.1 Network Model

The approach that we propose is based on the same philosophy used in technologies like CSFQ [14] and Corelite [12]. The network's edge routers maintain per-flow state information, and label packets based on rate at which flows send packets. Core routers maintain no per-flow state. Forwarding decisions are based on the labels that packet carry and aggregate state information such as queue length. Thus the edge and the core routers work in tandem to provide per-flow allocations. We build on these principles to provide resource allocation based on utility functions.

2.2 Network Objective

Let us assume that all flows provide the network their utility functions. There are a variety of objective functions that can be used to accomplish different goals. In the following discussion we consider two possible objectives, provide the intuition behind them, and motivate our choice of one of them as an objective that we use in this paper.

A possible network objective is to maximize the aggregate utility at every link in the network. i.e., at every link in the network, maximize $\sum_{i=1}^{M} U_i(r_i)$, subject to the constraint $\sum_{i=1}^{M} r_i \leq C$, where M is the number of flows sharing the link, $U_i(r_i)$ is the utility derived by flow i for a allocation r_i and C is the total link capacity. Another potential objective is to maximize the aggregate system utility, i.e, maximize $\sum_{i=1}^{N} U_i(r_i)$, where N is the total number of flows in the network. For a network with just a single link both the objectives are identical. However, for a multi-hop network they are different. For instance, consider the example shown in Figure 2. Here, $f1$ is a high priority flow and hence has a larger incremental utility than that for $f2$ and $f3$. If the available bandwidth is two units, then if we use the first objective function we will allocate both units to $f1$ in both the links. This maximizes the utility at every link in the network(3.0 units at every link) and the resultant system utility is 3.0 because only $f1$ received a bandwidth allocation. However, if we use the second objective function, we will allocate two units to $f2$ and $f3$ in each of the links. Though the aggregate utility at any given link is only 2.0, the resultant aggregate system utility is 4.0. This difference in allocation results from different interpretations

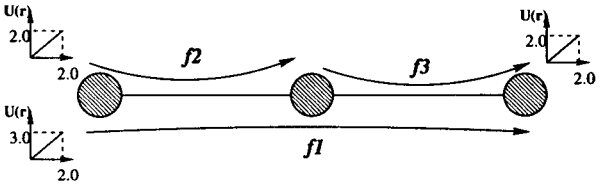


Fig. 2. Bandwidth Allocation Example

of the utility function. The first objective function treats utility functions from a user's perspective by collapsing the entire network into a single unified resource, neglecting the hop count. Thus, this interpretation has 2 key characteristics: (i) it is topology agnostic, i.e. a user does not have to be concerned with how many hops a flow traverses when specifying a utility value, and (ii) it maintains the relative importance of different flows as specified by the utility function across all links.

While the first objective function suites an user's perspective the second treats them from a resource pricing point of view. In this interpretation, the more hops a flow traverses the more resources the flow utilizes and the more a user should pay for comparable performance. Specifically, the utility functions can be viewed as quantifying a user's willingness to pay. Optimizing the second objective function can maximize the network operator's revenue. Networks which employ such an optimization criterion require a user to be cognizant of the hop count of the end to end path traversed by his flow and alter the utility function to get performance comparable to a case with a different number of hops.

Arguably, a case can be made in favor of either of the cases mentioned here or many other possible objectives. As our focus in this paper is to view utility function as a guide to user satisfaction, independent of network topology, we choose to focus on the former objective function.

3 Distributed Framework and Algorithms

In this section, we describe the distributed framework that provides rate allocations that approximate the desired network objective. A key characteristic of the framework is that only the routers at the edges of the network maintain per-flow state information and have access to the utility function of the flows. The core routers however, treat packets independent of each other. They do not perform any per-flow processing and have a simple forwarding behavior.

The framework has two primary concepts: First, an ingress edge router logically partitions a flow into *substreams*. The substreams correspond to different slopes in the utility function of the flow. Substreaming is done by appropriately labeling the headers of packets using incremental utilities. Second, a core router has no notion of a flow, and treats packets independent of each other. The forwarding decision at any router is solely based on the incremental utility labels on the packet headers. Routers do not drop a packet with a higher incremental utility label as long as a lower priority packet can instead be dropped. In other words, the core router attempts to provide the same forwarding behavior of a switch implementing a multi-priority queue by using instead a simple FIFO scheduling mechanism, eliminating any need for maintaining multiple queues or sorting the queue. For ease of explanation, in this paper, we describe the algorithms in the context of utility function U4 in Figure 1[2].

[2] Many utility functions such as U1 can be easily approximated to a piece-wise function similar U4. For functions such as U3 we are still working on appropriate labeling algorithms that provide the right allocation with least amount of oscillations

3.1 Substreaming at the Edge

Every ingress edge router maintains the utility function, $U(r)$, and the current sending rate, r, corresponding to every flow that it serves. The current sending rate of a flow can be estimated using an algorithm similar to the one described in CSFQ [14]. The edge router then uses a labeling algorithm to compute an incremental utility value, u_i, that should be marked on the packet header. The result of this procedure is that the flow is logically divided into k substreams of different incremental utilities, where k is the number of regions or steps[3] in the utility function from 0 to r. The u_i field is set to $(U(r_j) - U(r_{j-1}))/(r_j - r_{j-1})$ which represents the increment in utility that a flow derives per unit of bandwidth allocated to it, in the range (r_j, r_{j-1}). Thus all packets have a small piece of information based on the utility function of the flow embedded in them.

3.2 Maximizing Aggregate Utility

Routers accept packets such that a packet with a higher incremental utility value is not dropped as long as a packet with a lower incremental utility could instead be dropped. Such a policy ensures that in any given router, the sum of u_i of the accepted packets is maximized. There are many different ways by which such a dropping policy can be implemented in the router.

One solution is to maintain a queue in the decreasing order of priorities[4]. When the queue size reaches its maximum limit, q_{lim}, the lowest priority packet in the queue can readily be dropped and incoming packet can be inserted appropriately. This would provide the ideal result. But in a high speed router, even with a moderate queue size, such a solution will be inefficient as the processing time allowable for any given packet will be very small. In the following section, we propose an algorithm that approximates the behavior of such a dropping discipline using a simple FIFO queue, without the requirements of maintaining packets in a sorted order or managing per-flow or per-class information.

Priority Dropping with a FIFO Queue: The problem of dropping packets with lower incremental utility labels before packets with a higher incremental utility can be approximated to the problem of dynamically computing a minimum threshold value that a packet's label must have, in order for a router to forward it. We call this value the threshold utility, u_t. We define threshold utility as the minimum incremental utility that a packet must have for it to be accepted by the router. The two key constraints on u_t are that it must be maintained at a value which will (a) result in enough packets being accepted to fully utilize the link and (b) not cause buffer overflow at the router.

In Figure 3, $R(u)$ is a monotonically decreasing function of the incremental utility u. It represents the cumulative rate of all packets that are forwarded

[3] A step in refers to a contiguous region of resource values with the same slope

[4] This will be in addition to the FIFO queue, that is required to avoid any reordering of packets.

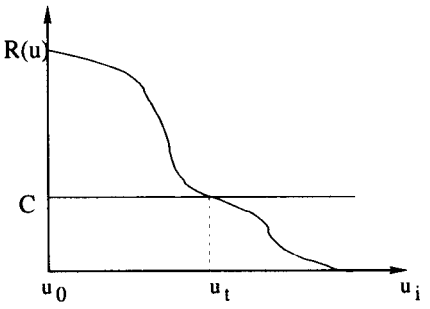


Fig. 3. Threshold Utility

through a link for a given threshold utility value, u_j. So $R(u_j) = \sum_{k=j}^{max} r(u_k)$, where $r(u_k)$ is the rate of packets entering an output link with an incremental utility label of u_k. The threshold utility, u_t is a value which satisfies the condition $R(u_t) = C$, where C is the capacity of the output link. Note that for a given $R(u)$, there may not exist a solution to $R(u) = C$ because of discontinuities in $R(u)$. Also the function $R(u)$ changes with time as flows with different utility functions enter and leave the system and when existing flows change their sending rates. Hence, tracking the function is not only expensive but may in fact be impossible. So in theory, an algorithm that uses the value of a threshold utility for making accept or drop decisions, cannot hope to replicate the result obtained by an approach that involves sorting using per-flow state information. Our objective is to obtain a reasonably close approximation of the threshold utility so that the sum of utilities of flows serviced by the link closely tracks the optimal value, while the capacity of the output link is fully utilized.

First, we give the intuition behind the algorithm that a router uses to maintain the threshold utility u_t for an output link and then provide the pseudo code for the algorithm. We then describe how it is used to make the forward or drop decision on a new incoming packet.

The objectives of the algorithm are (i) to maintain the value of u_t such that for the given link capacity the sum of the utilities of all the accepted packets is close to the maximum value possible, and (ii) to maintain the queue length around a specified lower and upper threshold values(q_{lth} and q_{uth}). There are three key components in the algorithm. (a) to decide whether to increase, decrease or maintain the current value of u_t, (b) to compute the quantum of change and (c) to decide how often u_t should be changed. The key factors that determine these decisions are avg_qlen, an average value of the queue length computed (well known methods for computing the average, like the exponential averaging technique can be used for this purpose) and q_{dif}, the difference between the virtual queue length value at the current time and when the threshold u_t was last updated. The virtual queue length is a value that is increased on an enqueue event by the size of the packet received if its $u_i \geq u_t$. The value is decreased by the size of the packet either on a deque or during a successful enque of a packet

with a label less than than the u_t[5]. The latter enables corrective action when packets are being dropped due to a incorrect(large) threshold. Thus, the virtual queue length is simply a value that increases and decreases without the physical constraints of a real queue. Maintaining a virtual queue length in this manner provides an accurate estimate of the state of congestion in the system. Note that even when the real queue overflows, the virtual queue length will increase, resulting in a positive q_{dif} reflecting the level of congestion. Similarly, when the u_t value is very large and no packets are being accepted, the virtual queue length will decrease, resulting in a negative value of q_{dif}. q_{dif} reflects the rate at which the length of the virtual queue is changing. When there is a sudden change in $R(u)$, q_{dif} provides a early warning signal which indicates that u_t may need to be modified. However, if the link state changes from uncongested to congested slowly, the absolute value of q_{dif} may remain small. But a value of avg_qlen that is beyond the specified queue thresholds indicates that u_t needs to be changed.

The quantum of change applied to u_t is based on the amount of buffer space left – given by the queue length, and the rate at which the system is changing – given by q_{dif}. Congestion build up, is equivalent to $R(u)$ in Fig. 3 shifting to the right. To increase the threshold, we use a heuristic to determine a target value u_{itgt} such that $R(u_{itgt}) < C$. This is used to significantly reduce the probability of tail drops. Currently this value of u_{itgt} is based on the average u_i values of all the accepted packets and the maximum u_i value seen in the last epoch. The value of u_t is then incremented in step sizes that are based on the estimated amount of time left before the buffer overflows. Similar computation is done to decrease the threshold where u_{dtgt} is based on the average u_i values of all packets dropped in the last epoch. The pseudocode for updating the threshold is as follows:

$$
\begin{aligned}
&\text{if } (avg\_qlen < q_{lth}) or (q_{dif} < -K_q) \\
&\quad time\_left = avg\_qlen / q_{dif} \\
&\quad change = (u_t - u_{dtgt}) / time_left \\
&\text{else if } (avg\_qlen > q_{uth}) or (q_{dif} > K_q) \\
&\quad time\_left = (q_{lim} - avg\_qlen) / q_{dif} \\
&\quad change = (u_{itgt} - u_t) / time\_left \\
&u\_t += change
\end{aligned}
$$

There are two events that trigger a call to the update-threshold() function. They are (a) whenever $|q_{dif}| > K_q$ and (b) a periodic call at the end of a fixed size epoch. The first trigger ensures fast reaction. The value of K_q is a configurable parameter and is set such that we do not misinterpret a typical packet burst as congestion. Also, it provides a self-healing feedback loop. For instance, when congestion is receding, if we decrease u_t by steps that are smaller than optimal, this trigger will result in the change being applied more often. Case (b) ensures that during steady state, the value of u_t is adjusted so that the queue length is maintained within the specified queue thresholds.

[5] This case would occur when link is not congested but u_t is incorrectly large.

The forwarding algorithm is very simple. When the link is in a congested state($avg\_qlen > q_{lth}$), if $u_i \leq u_t$ the packet is dropped. Otherwise the packet is accepted.

3.3 Variants of SURF

The framework described above is flexible and can be tailored to specific needs by choosing appropriate utility functions. The labeling algorithms at the edge router can be tailored to label the incremental utilities for specific cases. The forwarding and threshold computation algorithms remain the same. This is a big advantage. In this section, we describe a few specific cases of the edge labeling algorithm and provide the pseudo-code.

Fair Bandwidth Allocation. A common notion of fair bandwidth allocation is one in which all flows that are bottle necked at a link get the same rate, called the fair share rate. To achieve such an allocation, all we need to do is assign identical utility functions with constantly decreasing incremental utilities to all flows. For ease of understanding we provide a labeling procedure for an idealized bit-by-bit fluid model. Let u_{max} be the maximum possible value of incremental utility.

label(pkt)
  $served+ = 1$
  $pkt.u_i = u_{max} - served$

where the value of *served* is reset to 0, after a fixed size labeling epoch, say 1 sec. Let us suppose that the rate at which each flow is sending bits is constant. The result of this labeling algorithm then is that during any given second, the bits from a flow sending at rate r bits per second are marked sequentially from u_{max} to $u_{max} - r$. The router in a bottleneck link will compute the threshold u_t and drop packets from all flows with $u_i < u_t$. This results in fair bandwidth allocation. This is an alternate implementation of CSFQ [14]. As we will see in the next section, a key advantage of this approach is that it allows us to convey rate information as well as intra-flow utility using the same field in the packet header.

Intra-flow Priorities. Consider a flow, i, which is sending packets with multiple priority levels at a cumulative rate r_i. For instance, the levels could be I, P and B frames in an MPEG video stream or layers in a layered multicast stream [9]. The utility function corresponding to the flow will be similar to U4 in Figure 1. Independent of the number of layers and rate allocated to other flows, if flow i's packets need to dropped, we would like the packets from layer $j+1$ to be dropped before layer j. To achieve such a dropping behavior, the end hosts must communicate the relative priority of a packet to the edge router. A simple mechanism to accomplish this could be in the form of a field in the packet header. The desired dropping behavior honoring intra-flow drop priorities can be

achieved in the proposed framework by using a labeling procedure similar to the pseudo-code given below. The forwarding and threshold computation algorithms remain unchanged.

label(pkt)
  $p = pkt.intraflow\_priority$
  $served[p] += pkt.size$
  $cum\_rate[p] = cum\_rate[p-1] + est\_rate[p]$
  if ($served[p] < cum\_rate[p-1]$) or
    ($served[p] \geq cum\_rate[p]$)
      $served[p] = cum\_rate[p-1]$
  $pkt.u_i = u(served[p])$

Figure 4 describes the above pseudo-code. In the code given above, $est\_rate[p]$ is the estimated rate at which a flow is sending packets of a certain priority level p and $cum\_rate[p]$ is simply $\sum_{i=1}^{p} est\_rate[p]$(assuming 1 to be the highest priority level). $est\_rate[p]$ can be computed using a method similar to the one used in [14]. $served[p]$ maps the packet received onto the appropriate region in its utility function. $u(r)$ gives the incremental utility of the region corresponding to rate r.

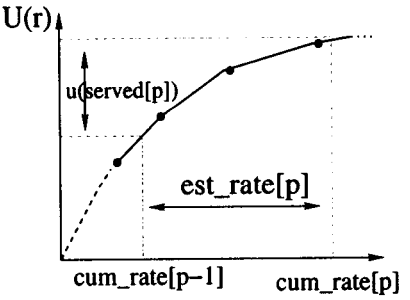


Fig. 4. Labeling Algorithm

Improving TCP Performance. The labeling algorithm used by the edge router can be tailored to improve the performance of TCP. Specifically, it can mitigate two primary causes of a reduction in throughput – (i) drop of a retransmitted packet (ii) back to back drops of multiple packets. To accomplish the former, the labeling procedure can label retransmitted packets with the highest allowable incremental utility for the flow; and to accomplish the latter, the labeling algorithm can assign consecutive packets to different priority levels thereby reducing the chances of back-to-back drops. Such an implicit assignment of interleaved priorities is also useful for audio streams, whose perceived quality improves when back-to-back drops are avoided.