# Coexistence of QoS and Best-Effort Flows — Routing and Scheduling [*]

Klara Nahrstedt, Shigang Chen
Department of Computer Science
University of Illinois at Urbana-Champaign
{klara,s-chen5}@cs.uiuc.edu

## Abstract

The future high-speed networks will need to support diverse traffic and provide services to flows with Quality of Service (QoS) requirements as well as to best effort flows. In this paper we analyze the coexistence of the QoS and best effort flows from the routing and scheduling point of view. We concentrate in our routing and scheduling analysis on the *network bandwidth* resource. We present two sets of source routing algorithms: (1) the *bandwidth-constrained routing with imprecise state information* for QoS flows, and (2) the *maxmin fair routing* for best effort flows. The routing analysis includes an extensive description of various algorithms in their domains and their complexity discussion. Furthermore, we discuss *two level hierarchical scheduling* which is tailored towards the needs raised by the coexistence of QoS and best effort flows. We show that this scheduling design accomplishes two design goals (1) guaranteeing QoS for QoS flows and (2) ensuring fairness for best effort flows, and the overhead is reasonably small and comparable with the time complexity of the single-level fair queuing scheduling.

## 1 Introduction

The future high-speed networks will carry many concurrent flows with diverse requirements. Hence, it is crucial that the network bandwidth and other network resources are shared effectively and fairly among all competing flows. In this paper we will consider two sets of flows: *QoS flows* which post QoS requirements on the established end-to-end path, and *best effort flows* which do not have any specific QoS requirements on the established end-to-end path. There exist numerous network services which need to be examined and revisited when coexistence of these flows is studied. In this paper we will analyze two network services: *Routing* and *Scheduling*. In our analysis we will concentrate on *network bandwidth* resource.

### 1.1 Routing

In general, the problem of routing is difficult due to a number of reasons.

- First, distributed applications such as teleconference, video-on-demand, Internet phone and web-based games have very diverse QoS constraints on delay, delay jitter, lose ratio, bandwidth, etc. Multiple constraints often make the QoS routing problem intractable. In particular, finding the least-cost path with one path constraint or finding a feasible path with two independent path constraints is NP-complete [11].

- Second, any future integrated-service network will carry both QoS traffic and best-effort traffic, which makes the issue of performance optimization complicated. A primary task of routing is to maximize the *resource efficiency*, which is measured by two goals. One goal is to maximize the number of QoS flows that are admitted into the network, which is equivalent to minimize the *call-blocking ratio*. The other goal is to optimize the throughput and responsiveness of best-effort traffic. The two goals may contradict each other. That is because (1) the first goal considers only QoS traffic, (2) the second goal considers only best-effort traffic and (3) however the two types of traffic may have very different distributions. Generally speaking, it is hard to determine the best operating point for both types of traffic if their

[*]This work was supported by the ARPA grant under contract number F30602-97-2-0121 and the National Science Foundation Career grant under contract number NSF CCR 96-23867.

distributions are independent. Although the QoS traffic will not be affected by the best-effort traffic due to resource reservation, the throughput of the best-effort traffic will suffer if the overall traffic is misjudged. For example, links with light QoS traffic may have heavy best-effort traffic, and by many QoS routing algorithms, these links are often considered as good candidates for new QoS flows, which however causes the already congested best-effort traffic even more congested.

- Third, the network state changes dynamically due to transient load fluctuation, connections in and out, and links up and down. The growing network size makes it increasingly difficult to gather up-to-date state information in a dynamic environment, particularly when wireless communication is involved. The performance of a QoS routing algorithm can be seriously degraded if the state information being used is outdated.

In this paper we present two sets of source routing algorithms: (1) the *bandwidth-constrained routing with imprecise state information* for QoS flows, and (2) the *maxmin fair routing* for best effort flows. The goal of our bandwidth-constrained routing algorithm with imprecise state informations is to maximize the probability of success in finding a feasible path for a new QoS flow in a dynamic network environment. The goal of our maxmin fair routing algorithm is to assign routes to new best effort flows such that the performance of the maxmin bandwidth allocation can be maximized. The routing analysis includes an extensive description of various algorithms in their domains and their complexity discussion.

### 1.2 Scheduling

Scheduling of flows with diverse network requirements can be done by a single-level weighed fair queuing algorithms. However, the complexity of this algorithm increases with the number of flows and the book-keeping of flow states becomes more complex. Hence, a hierarchical scheduling is more appropriate to deploy as it scales better towards the increasing number of flows with different network requirements. We modified the hierarchical fair queuing scheduling algorithm proposed by Bennett and Zhang [2] and tuned it towards specific needs raised by the co-existence of QoS flows and best effort flows.

In this paper we discuss the *two-level hierarchical scheduling.* The top scheduler schedules a QoS server and best-effort server which enforce the bandwidth QoS requirements for QoS flows and ensure fairness for best effort flows. Both servers as well as the top level scheduler use weighted fair queuing for bandwidth allocation. We show that this scheduler design accomplishes the two design goals (guaranteeing QoS for QoS flows and ensuring fairness for best effort flows), and the overhead is reasonably small and comparable with the time complexity of the single-level fair queuing scheduling.

### 1.3 Outline

The core of the paper is divided into four sections. The Section 2 presents the network model and specifies the notation for the bandwidth partition between the QoS and best effort flows. Section 3 describes bandwidth-constrained routing algorithms using the imprecise state model. Section 4 analyses the best effort maxmin routing algorithm which uses the fairness-throughput relation for route finding and allocation. Furthermore, we will discuss an approximation algorithm which decreases the communication overhead for collecting the routes of all flows. Section 5 proposes an integrated hierarchical packet scheduling which enforces network bandwidth for both QoS and best effort flows. The paper concludes with Section 6.

## 2 Model

A network is modeled as a set $V$ of nodes that are interconnected by a set $E$ of full-duplex, directed communication links. Let $F$ be the set of flows in the network. We study connection-oriented networks where each flow has a fixed source (destination) and is assigned a fixed route through which all packets of that flow are transmitted in the FIFO order [5, 13, 12]. For a flow $f \in F$, the set of links on its route is denoted as $L(f)$. The set of flows through a link $l$ is denoted as $F(l)$.

We study two types of flows in this paper. A *QoS flow* $f$ has a bandwidth requirement $B(f)$, which must be guaranteed (reserved for $f$ on each link in $L(f)$) in order to ensure an acceptable quality. An example is an audio session between two remote users. The set of QoS flows in the network is denoted as $F_{qos}$. The set of QoS flows through a link $l$ is denoted as $F_{qos}(l)$.

A *best-effort flow* can operate at any bandwidth level, and hence the reservation of bandwidth is not needed. Examples are file transmission ($ftp$), web-page download and database retrieval. The set of best-effort flows in the network is denoted as $F_{best}$. The set of best-effort flows through a link $l$ is denoted as $F_{best}(l)$. We have $F_{qos} + F_{best} = F$ and $F_{qos}(l) + F_{best}(l) = F(l)$.

Each link $l$ has a bandwidth capacity $C(l)$, among which the part reserved by the QoS flows is denoted as $C_{qos}(l)$ and the part available to the best-effort flows is denoted as $C_{best}(l)$. $C_{qos}(l) = \Sigma_{f \in F_{qos}(l)} B(f)$, and $C_{best}(l) = C(l) - C_{qos}(l)$. It is clear that the values of $C_{qos}(l)$ and $C_{best}(l)$ are not fixed. When a new QoS flow $f$ is routed through $l$, $C_{qos}(l)$ is increased by $B(f)$ and $C_{best}(l)$ is decreased by the same amount. In order to prevent the best-effort flows from being starved, $C_{qos}(l)$ is upper-bounded by $\lambda C(l)$, where $\lambda$ is a system parameter less than 1. For every link $l$, the condition $C_{qos}(l) \leq \lambda C(l)$ must *always* be satisfied. Hence, the bandwidth for best-effort flows is at least $(1 - \lambda)C(l)$.

A new QoS flow $f$ with $B(f)$ requirement can be accepted by a link $l$ only if $\lambda C(l) - C_{qos}(l) \geq B(f)$. $\lambda C(l) - C_{qos}(l)$ is called the *residual* bandwidth of $l$ and is denoted as $bandwidth(l)$. A simple path $p$ consists of a set of connected links. The residual bandwidth of a path is defined as

$$bandwidth(p) = \min_{l \in p}\{bandwidth(l)\}$$

| | |
|---|---|
| $\langle V, E \rangle$ | the set $V$ of nodes and the set $E$ of links in the network |
| $F$ | the set of flows in the network |
| $F_{qos}$ | the set of QoS flows in the network |
| $F_{best}$ | the set of best-effort flows in the network |
| $F(l)$ | the set of flows through a link $l$, $F(l) \subseteq F$ |
| $F_{qos}(l)$ | the set of QoS flows through a link $l$ |
| $F_{best}(l)$ | the set of best-effort flows through a link $l$ |
| $L(f)$ | the set of links on the route of a flow $f$ |
| $B(f)$ | the bandwidth requirement of a QoS flow $f$ |
| $C(l)$ | the capacity of a link $l$ |
| $C_{qos}(l)$ | the bandwidth reserved by the QoS flows in $F_{qos}(l)$ |
| $\lambda\, C(l)$ | the maximum bandwidth that can be reserved by the QoS flows, $C_{qos}(l) \leq \lambda\, C(l)$ |
| $C_{best}(l)$ | the bandwidth available to the best-effort flows in $F_{best}(l)$, $C_{best}(l) = C(l) - C_{qos}(l)$ |
| $bandwidth(l)$ | the residual bandwidth of a link $l$, $bandwidth(l) = \lambda\, C(l) - C_{qos}(l)$ |

Table 1: Notations

## 3 QoS Routing

### 3.1 Bandwidth-constrained routing and imprecise state information

We study the *bandwidth-constrained routing* problem, i.e., finding a path $p$ from $s$ to $t$ such that $bandwidth(p) \geq B$, where $s$, $t$ and $B$ are the source node, the destination node and the bandwidth requirement [1] of a new QoS flow, respectively. Any path from $s$ to $t$ satisfying the above constraint is called a *feasible* path.

We use the source routing strategy [7, 12, 14], in which every node maintains an image of the *global network state*, based on that a routing path is computed at the source node. However, the global state, which is typically maintained by a link-sate ( or distance-vector) protocol, is inherently *imprecise* in a dynamic network where the traffic load changes constantly. The imprecision is especially noticeable in large wide-area networks due to the following reasons. First, it takes non-negligible propagation delay for a local state change to be broadcasted to other nodes. Second, a link-state (or distance-vector) protocol updates the state information periodically or upon triggering when significant state change is detected. There exists a tradeoff between the update frequency and the overhead involved. For large scale networks, the excessive communication overhead often makes it impractical for the update frequency to be high enough to cope with the dynamics of network parameters such the bandwidth availability on every link. Third, the hierarchical approach is likely to be used to solve the scalability problem of routing in large networks [10]. However, the state aggregation in hierarchical routing increases the level of imprecision [12].

The imprecision of state information directly affects the routing performance. The goal of our algorithm is to maximize the probability of success in finding a feasible path in a dynamic network environment where the available information is imprecise. In the following, we shall first propose an imprecise state model, based on which the routing algorithm is described.

[1] We use $B$ as an abbreviation of $B(f)$ when only a single QoS flow is in discussion.

| | |
|---|---|
| $B_l$ | the estimated residual bandwidth of a link $l$ |
| $\Delta B_l$ | the estimated maximum change of $B_l$ before the next update period |

Table 2: Notations

## 3.2 Imprecise state model

Every node maintains a state vector which has an entry, denoted as $B_l$, for every link $l \in E$.

**1) Bandwidth:** $B_l$ keeps the residual bandwidth available on link $l$.

$B_l$ is updated periodically by a link-state protocol. It is inherently *imprecise* in a dynamic network as discussed in Section 3.1. We propose a simple imprecise state model which can be easily implemented. An additional state variable $\Delta B_l$ is required.

**2) Bandwidth Variation:** $\Delta B_l$ keeps the *estimated* maximum change of $B_l$ before the next update. The *actual* residual bandwidth of link $l$, denoted as *bandwidth*$(l)$, is expected to be between $B_l - \Delta B_l$ and $B_l + \Delta B_l$ in the next period.

In the following, we describe a possible way to calculate $\Delta B_l$. $\Delta B_l$ is updated periodically together with $B_l$. Consider an arbitrary update of $\Delta B_l$ and $B_l$. Let $\Delta B_l^{old}$ and $\Delta B_l^{new}$ be the values of $\Delta B_l$ before and after the update, respectively. Similarly, let $B_l^{old}$ and $B_l^{new}$ be the values of $B_l$ before and after the update, respectively. $B_l^{new}$ is provided by a link-state protocol. $\Delta B_l^{new}$ is calculated as follows.

$$\Delta B_l^{new} = \alpha \times \Delta B_l^{old} + (1-\alpha) \times |B_l^{new} - B_l^{old}|$$

The above formula is similar to the one used by TCP to estimate the round-trip delay. The factor $\alpha$ $(< 1)$ determines how fast the *history* information $(\Delta B_l^{old})$ is forgotten, and $(1-\alpha)$ determines how fast $\Delta B_l^{new}$ converges to $|B_l^{new} - B_l^{old}|$.

By the above formula, it is still possible for the actual residual bandwidth to be out of the range $[B_l - \Delta B_l, B_l + \Delta B_l]$. One way to make such probability negligible small is to enlarge $\Delta B_l$. Hence, we shall modify the formula and introduce another factor $\beta$ $(> 1)$.

$$\Delta B_l^{new} = \alpha \times \Delta B_l^{old} + (1-\alpha) \times \beta \times |B_l^{new} - B_l^{old}|$$

$\Delta B_l^{new}$ converges to $\beta \times |B_l^{new} - B_l^{old}|$ at a speed determined by $(1-\alpha)$.

The most related work was done by Guerin and Orda [12] and by Lorenz and Orda [14]. Their imprecision model is based on the probability distribution functions. For instance, every node maintains, for every link $l$, the probability $p_l(w)$ of link $l$ having a residual bandwidth of $w$ units, where $w$ ranges from zero to maximum possible value. The problem of how to maintain the probability distribution was not discussed. Our imprecision model is much simpler. For every link $l$, a node maintains two values, $B_l$ and $\Delta B_l$, from which the probability $p_l(w)$ can be derived, as shown in Section 3.3.

## 3.3 Routing algorithm

The purpose of our routing algorithm is to maximize the probability of success in finding a feasible path, given $B_l$ and $\Delta B_l$, $\forall l \in E$. An important part of the algorithm is to calculate the probability of a link $l$ satisfying a given bandwidth requirement. Such a probability is determined by the probability distribution function of *bandwidth*$(l)$. Let us consider the simple case. Assume that *bandwidth*$(l)$ is a random variable whose value is uniformly distributed in $[B_l - \Delta B_l, B_l + \Delta B_l]$. The probability density function of *bandwidth*$(l)$ is

$$f(x) = \begin{cases} \frac{1}{2\Delta B_l} & x \in [B_l - \Delta B_l, B_l + \Delta B_l] \\ 0 & x \notin [B_l - \Delta B_l, B_l + \Delta B_l] \end{cases}$$

Consider a new flow which source node, destination node and bandwidth requirement are $s$, $t$ and $B$, respectively. The probability of a link $l$ satisfying the bandwidth requirement is

$$\begin{aligned} & Pr(bandwidth(l) \geq B) \\ &= \int_B^{+\infty} f(x)\, dx \\ &= \begin{cases} \frac{B_l + \Delta B_l - B}{2\Delta B_l} & B \in [B_l - \Delta B_l, B_l + \Delta B_l] \\ 1 & B < B_l - \Delta B_l \\ 0 & B > B_l + \Delta B_l \end{cases} \end{aligned}$$

The probability of a path $p$ satisfying the requirement is

$$
\begin{aligned}
&Pr(bandwidth(p) \geq B) \\
&= \prod_{l \in p} Pr(bandwidth(l) \geq B) \\
&= \begin{cases} \prod_{l \in p} \frac{min\{B_l + \Delta B_l - B, 2\Delta B_l\}}{2\Delta B_l} & \forall l \in p, B < B_l + \Delta B_l \\ 0 & \exists l \in p, B \geq B_l + \Delta B_l \end{cases}
\end{aligned}
$$

The routing algorithm is designed to find a path $p$ from $s$ to $t$ which maximizes $Pr(bandwidth(p) \geq B)$.

**Algorithm 1.**

1. Let $E' = \{l \mid B < B_l + \Delta B_l, l \in E\}$. Remove all links in $E - E'$, and the resulting graph is denoted as $\langle V, E' \rangle$. If there does not exist a path from $s$ to $t$ in $\langle V, E' \rangle$, reject the flow and return.

2. $\forall l \in E'$, a weight $w_l = -log \frac{min\{B_l + \Delta B_l - B, 2\Delta B_l\}}{2\Delta B_l}$ is assigned.

3. Use the Dijstra's shortest path algorithm to find the *least-weighted* [2] path $p$ from $s$ to $t$ in $\langle V, E' \rangle$.

4. Select $p$ as the routing path. If the flow is successfully established through $p$, return. Otherwise, reject the flow and return.

We can also assume other distribution functions for $bandwidth(l)$. A more generic routing algorithm is presented below:

**Algorithm 2.**

1. Let $E' = \{l \mid Pr(bandwidth(l) \geq B) > 0, l \in E\}$. Remove all links in $E - E'$, and the resulting graph is denoted as $\langle V, E' \rangle$. If there does not exist a path from $s$ to $t$ in $\langle V, E' \rangle$, reject the flow and return.

2. $\forall l \in E'$, a weight $w_l = -log\ Pr(bandwidth(l) \geq B)$ is assigned.

3. Use the Dijstra's shortest path algorithm to find the *least-weighted* path $p$ from $s$ to $t$ in $\langle V, E' \rangle$.

4. Select $p$ as the routing path. If the flow is successfully established through $p$, return. Otherwise, reject the flow and return.

where the value of $Pr(bandwidth(l) \geq B), l \in E$ is determined by the probability distribution function of $bandwidth(l)$.

[2]The least-weighted path $p$ is the one which minimizes $\sum_{l \in p} w_l$.

The design goal of Algorithm 2 is to maximize the probability for the selected path to satisfy the bandwidth requirement. It does not consider other optimization criteria such as the path length. When there exist many paths that satisfy the QoS requirement, the shortest path in terms of *number of hops* is often desired due to a better statistical performance [15]. Algorithm 3 tries to take the path length into account. Let $d_{s,t}$ be the distance between the source $s$ and the destination $t$.

**Algorithm 3**

1. Let $E' = \{l \mid Pr(bandwidth(l) \geq B) > 0, l \in E\}$. Remove all links in $E - E'$, and the resulting graph is denoted as $\langle V, E' \rangle$. If there does not exist a path from $s$ to $t$ in $\langle V, E' \rangle$, reject the flow and return.

2. $\forall l \in E$, a weight $w_l = -log\ Pr(bandwidth(l) \geq B)$ is assigned.

3. Run the Bellman-Ford algorithm, which consists of $|V| - 1$ iterations.

   (a) After the $d_{s,t}$th iteration, it finds the least-weighted path $P_0$ from $s$ to $t$ with a length of $d_{s,t}$.

   (b) After the $(d_{s,t} + 1)$th iteration, it finds the least-weighted path $P_1$ from $s$ to $t$ with a length of no more than $d_{s,t} + 1$.

   (c) After the $(|V| - 1)$th iteration, it finds the least-weighted path $P_2$ among all simple paths from $s$ to $t$.

4. Select $P_0$ as the routing path. If the flow is successfully established through $P_0$, return. Otherwise, go to the next step.

5. If $P_1 = P_0$, go to the next step. Otherwise, select $P_1$ as the routing path. If the flow is successfully established through $P_1$, return. Otherwise, go to the next step.

6. Select $P_2$ as the routing path. If the flow is successfully established through $P_2$, return. Otherwise, reject the flow and return.

## 4 Best-Effort Routing

For the best-effort flows, we shall first review the *maxmin bandwidth allocation*, based on which a new routing policy, called the *maxmin fair routing*, is defined. Throughout this section, we call $C_{best}(l)$ the *available bandwidth (capacity)* for best effort flows of link $l$.

## 4.1 Maxmin Bandwidth Allocation

### 4.1.1 Brief Review

The maxmin allocation was first proposed by Jaffe [9] as a flow control technique which distributes the network bandwidth fairly among the best-effort flows. Much further research [1, 3, 4, 5, 17] has been done since then. It has been accepted by the ATM Forum as one of the traffic management approaches for the ABR(Available Bit Rate) service.

Its name comes from the fact that the maxmin allocation always *maximizes* the bandwidth allocated to those flows that receive the *minimum* bandwidth among all flows. The maxmin allocation has two basic properties:

1. *Fairness property:* At each link $l$, every flow $f \in F_{best}(l)$ is allocated an equal share of the available bandwidth $C_{best}(l)$. However, if $f$ receives a lower bandwidth at another link on its route, the bandwidth of $f$ is allocated according to the bandwidth allocation at the bottleneck link on its route.

2. *Maximum throughput property:* The entire available bandwidth $C_{best}(l)$ must be allocated to the flows in $F_{best}(l)$ unless every flow $f$ in $F_{best}(l)$ has a bottleneck link elsewhere on its route which limits the maximum bandwidth $f$ can receive.

The maxmin bandwidth of each best-effort flow is determined by the bottleneck link on its route. A global bottleneck based algorithm which assigns the maxmin bandwidth to each best-effort flow was described in [5, 16] and is briefly summarized below. A distributed algorithm was given in [9].

A global bottleneck link $l_b$ is defined as the link which has the smallest bandwidth per flow, $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$. Since $l_b$ is the most congested link in the network, it is the bottleneck link for each flow $f \in F_{best}(l_b)$. We allocate an equal share of the available bandwidth, i.e. $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$, to each $f$. Then all flows in $F_{best}(l_b)$ are removed from the network. The available bandwidth of every affected link is reduced by the amount consumed by the removed flows: For each $f \in F_{best}(l_b)$, the available bandwidth of every link on the route of $f$ is reduced by $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$. After that, the above procedure is repeated until every flow is assigned a bandwidth and removed from the network.

### 4.1.2 Fairness-Throughput Optimality

We formalize an important property for the maxmin allocation. A *feasible bandwidth allocation* $\Psi : F_{best} \rightarrow R^+$ is a function which satisfies the following condition

$$\forall l \in E, \sum_{f \in F_{best}(l)} \Psi(f) \leq C_{best}(l)$$

Let $\Psi(F_{best})$ be the *list* of values $(\Psi(f) \mid \forall f \in F_{best})$ in the increasing order. Note that in mathematics $\Psi(F_{best})$ normally represents a *set* of values $\{\Psi(f) | f \in F_{best}\}$. In this paper we make a different interpretation by introducing an increasing order to $\Psi(F_{best})$ and using it as an ordered *list*. A link $l$ is said to be *saturated* by $\Psi$ if

$$\sum_{f \in F_{best}(l)} \Psi(f) = C_{best}(l)$$

**Definition 1** *Given two feasible bandwidth allocations $\Psi$ and $\Psi'$ on $F_{best}$, we define the fairness-throughput relations: (1) $\Psi(F_{best}) = \Psi'(F_{best})$ if the two lists are identical, and (2) $\Psi(F_{best}) > \Psi'(F_{best})$ if there exists a prefix of $\Psi(F_{best})$, $(b_1, b_2, ..., b_i)$, and a prefix of $\Psi'(F_{best})$, $(b'_1, b'_2, ..., b'_i)$, such that $b_i > b'_i$ and $b_j = b'_j, 1 \leq j \leq i-1$.*

The above relations place a total order on the set of all feasible allocations. The ordering is based on two performance measurements, *fairness* and *throughput*. In more descriptive and less precise words, an allocation is larger than the other if it is fairer and/or generates more throughput, which is illustrated by Figure 1, where three best-effort flows, $f_1$, $f_2$ and $f_3$, share two links with available bandwidths 8 and 10, respectively. Consider three different allocations, $\Psi_1$, $\Psi_2$ and $\Psi_3$. By Definition 1, $\Psi_1(F_{best}) > \Psi_2(F_{best})$, The reason is that $\Psi_1$ is fairer as it splits the available bandwidth of link $l_1$ equally between $f_1$ and $f_2$ and thus maximizes the smallest element in the list of $\Psi_1(F_{best})$. $\Psi_3$ is also fair. However, it does not fully utilize the capacity of link $l_2$. Hence, $\Psi_1(F_{best}) > \Psi_3(F_{best})$.

Fairness and throughput are often conflicting measurements. For example, $\Psi_2(F_{best})$ has more overall

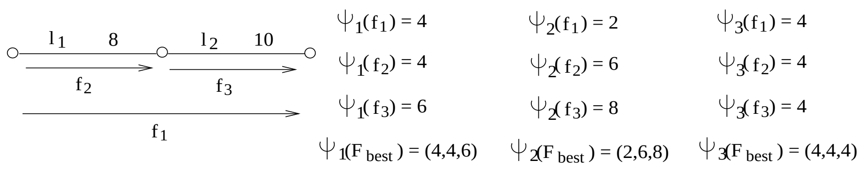

Figure 1: Consider three different allocations, $\Psi_1(F_{best}) = (4,4,6)$, $\Psi_2(F_{best}) = (2,6,8)$, and $\Psi_3(F_{best}) = (4,4,4)$, among which $\Psi_1$ is the maxmin fair allocation. $\Psi_1(F_{best}) > \Psi_2(F_{best})$ because it is fairer. $\Psi_1(F_{best}) > \Psi_3(F_{best})$ because it generates more overall throughput.

throughput [3] but $\Psi_1(F_{best})$ is fairer between $f_1$ and $f_2$. The fairness-throughput relations defined in Definition 1 evaluate an allocation based on a measurement which provides a tradeoff between the fairness and the overall throughput. We shall establish a theorem showing that the maxmin allocation maximizes the fairness-throughput performance, i.e. $\Psi_m(F_{best}) \geq \Psi(F_{best})$, where $\Psi$ is an arbitrary feasible allocation and $\Psi_m$ is the notation specially for the maxmin allocation here and in the rest of the paper. The proof of the following theorem can be found in [6].

**Theorem 1** *$\Psi_m(F_{best}) \geq \Psi(F_{best})$, for any feasible allocation $\Psi$.*

## 4.2 Maxmin Routing

In this section, the optimization property of the maxmin bandwidth allocation illustrated in Theorem 1 is extended from the domain of flow control to the domain of routing. We first define the concept of *maxmin routing.*

### 4.2.1 Definition of Maxmin Routing

Let $\langle V, E\rangle$ be a network where bandwidth is alway allocated to flows by maxmin. Let $F_{best}$ be the set of existing best-effort flows, each of which has a fixed route. Consider a new best-effort flow $f_0$. The task of routing is to assign a route $r$ to $f_0$. Each different route for $f_0$ results in a different maxmin bandwidth allocation $\Psi_{m,r}$ on $F_{best} \bigcup \{f_0\}$. The purpose of *maxmin routing* is to find a route $r$ such that $\Psi_{m,r}(F_{best} \bigcup \{f_0\}) \geq \Psi_{m,r'}(F_{best} \bigcup \{f_0\})$, for any feasible route $r'$ of $f_0$.

The maxmin routing is a new problem which is different from the maxmin allocation studied by the previous publications. The problem solved by the latter is as follows: given a network and a set of flows with fixed routes, how to assign the network bandwidth to the flows such that the network performance is optimized. The maxmin routing, however, assumes the network bandwidth is assigned based on the maxmin allocation. It then introduces another dimension, new flows. The problem to be solved is how to assign routes to new flows such that the performance of the maxmin allocation can be maximized.

### 4.2.2 Maxmin Routing Algorithm

The maxmin routing algorithm first adds $f_0$ to every link in the network [4] and then iteratively removes $f_0$ from the links which have the smallest per-flow bandwidth until the route for $f_0$ is found. By removing $f_0$ from the links with the smallest bandwidth per flow, the algorithm effectively routes $f_0$ around the most congested links and therefore maximizes the bandwidth allocated to the congested flows, which equals maximizing the low end of $\Psi_{m,r}$ and thus equals maximizing $\Psi_{m,r}$ because the low end of $\Psi_{m,r}$ is of more significance by definition.

The algorithm below consists of two phases. In the first phase, the bottleneck link of $f_0$ is found, which determines the maxmin bandwidth for $f_0$; in the second phase, the algorithm finds the rest of the route which maximizes $\Psi_{m,r}$. We mark links either green or red. Green links are candidates to form a route for $f_0$; red links are either not on any paths from the source to the destination or considered to be congested and thus rejected by the algorithm. A path consisting of only green links is called a *green path.* When we say "remove a best-effort flow $f$ from the network", we mean, $\forall l \in L(f), F_{best}(l) = F_{best}(l) - \{f\}$

[3]We define the *overall throughput* as the aggregate throughput of all flows.

[4]Note that we are not adding the flow to the links of the *real* network but to the data structure representing the network at a node doing the source routing.

and $C_{best}(l) = C_{best}(l) - \Psi_{m,r}(f)$, where $\Psi_{m,r}(f)$ is the bandwidth assigned to the flow by the algorithm. When we say the source (destination), we mean the source (destination) of $f_0$. The first phase of the algorithm is as follows.

1. For every link $l$ that is on a path from the source to the destination, add $f_0$ to $F_{best}(l)$ and mark $l$ as a green link. Mark other links red.
2. Find the global bottleneck link $l_b$ which has the smallest bandwidth per flow, $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$.
   (a) If $l_b$ is a red link, [5] then
      i. assign bandwidth $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$ to every flow in $F_{best}(l_b)$,
      ii. remove all flows in $F_{best}(l_b)$ as well as link $l_b$ from the network, and
      iii. repeat Step 2.
   (b) If $l_b$ is a green link and not all green paths from the source to the destination pass $l_b$, then
      i. remove $f_0$ from $F_{best}(l_b)$,
      ii. mark $l_b$ as a red link, and
      iii. repeat Step 2.
   (c) If $l_b$ is a green link and all green paths from the source to the destination pass $l_b$, then
      i. $l_b$ is the bottleneck link for $f_0$, and will be denoted as $l_0$ in the second phase,
      ii. assign bandwidth $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$ to every flow in $F_{best}(l_b)$ including $f_0$,
      iii. remove all flows in $F_{best}(l_b)$ except $f_0$ from the network, [6] and
      iv. go to the second phase of the algorithm.

Let $l_0$ be the bottleneck link of $f_0$, which is found at Step 2(c) in the first phase. Let $\Psi_{m,r}(f_0)$ be the bandwidth assigned to $f_0$. The second phase is to find the rest links which together with $l_0$ will form a route for $f_0$. It has the same control structure as the first phase except the subtle differences in the calculation of the global bottleneck and the treatment of $f_0$. The first phase focuses on finding the bottleneck link $l_0$ and determining $\Psi_{m,r}(f_0)$ whereas the second phase focuses on finding the rest of the route. Their conceptual differences make us decide to separate them, which also seems to make the algorithm more understandable. When the second phase terminates, $L(f_0)$ contains the links which form the route for $f_0$.

1. $L(f_0) = \{l_0\}$.
2. Among the links in $E - L(f_0)$, find the global bottleneck link $l_b$ which has the smallest bandwidth per flow. The bandwidth per flow of a link $l \in E - L(f_0)$ is calculated by $\frac{C_{best}(l) - \Psi_{m,r}(f_0)}{|F_{best}(l)|-1}$ if $l$ is a green link or $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$ if $l$ is a red link.
   (a) If $l_b$ is a red link, then
      i. assign bandwidth $\frac{C_{best}(l_b)}{|F_{best}(l_b)|}$ to every flow in $F_{best}(l_b)$,
      ii. remove all flows in $F_{best}(l_b)$ as well as link $l_b$ from the network,
      iii. repeat Step 2.
   (b) If $l_b$ is a green link and not all green paths from the source to the destination pass $l_b$, then
      i. remove $f_0$ from $F_{best}(l_b)$,
      ii. mark $l_b$ as a red link, and
      iii. repeat Step 2.
   (c) If $l_b$ is a green link and all green paths from the source to the destination pass $l_b$, then
      i. $L(f_0) = L(f_0) \bigcup \{l_b\}$,
      ii. assign bandwidth $\frac{C_{best}(l_b) - \Psi_{m,r}(f_0)}{|F_{best}(l_b)|-1}$ to every flow in $F_{best}(l_b)$ except $f_0$,
      iii. remove all flows in $F_{best}(l_b)$ except $f_0$ from the network, and
      iv. terminate if the links in $L(f_0)$ form a path from the source to the destination, or repeat Step 2 otherwise.

### 4.2.3 Discussion

The maxmin routing avoids the congested links by marking them red, and routes $f_0$ along those links whose per-flow bandwidth is as large as possible. Hence, it helps to improve the overall throughput of the network. However, in a long term, fairness may contradict throughput as the proposed algorithm may

[5] By the construction of the algorithm, $f_0 \notin F_{best}(l_b)$ if $l_b$ is a red link; $f_0 \in F_{best}(l_b)$ if $l_b$ is a green link.

[6] We can not remove $f_0$ from the network because the route of $f_0$, i.e. $L(f_0)$, is unknown.

select an excessively long path which reduces the overall bandwidth available for flows arriving successively. Research showed that short routing paths tend to yield high overall throughput [15]. The proposed algorithm can be modified to take the path length into consideration. A maximum allowable length is specified for a new flow based on the distance from the source to the destination. After marking a green link $l$ red, the algorithm tests whether there still exists a green path from the source to the destination whose length is bounded by the maximum allowable length. If the answer is false, the algorithm marks $l$ back green, selects the shortest green path from the source to the destination and then terminates.

## 4.3 An Approximation Algorithm

The proposed maxmin routing algorithm requires the knowledge of the routes of all existing flows. When flows join and leave the network frequently, the communication overhead for collecting the routes of all flows will be excessively high. Many routing algorithms [16, 19, 20] rely on the state information of the links in the network, instead of that of the flows. The state of the links can be collected and maintained at each node by the link-state algorithm [18], which has been implemented on many internetworks. In the following, we approximate the proposed maxmin routing algorithm by using a new state defined on each link.

Each link $l$ monitors the actual data rate $r(f)$ of every flow $f \in F_{best}(l)$.[7] By using the dynamic bandwidth allocation discussed in the next section, the actual data rate of a flow will approximate the expected maxmin bandwidth. The link $l$ keeps track of the set $F^b_{best}(l)$ of flows whose bottleneck links are $l$. How to maintain $F^b_{best}(l)$ is also discussed in the next section. Note that, according to the maxmin allocation, the rates of flows in $F^b_{best}(l)$ are higher than those of flows in $F_{best}(l) - F^b_{best}(l)$ whose bottleneck links are elsewhere on their routes other than $l$ [3]. A new state $r(l)$ is defined for $l$.

$$r(l) = max\{\frac{C_{best}(l) - \sum_{f \in F_{best}(l) - F^b_{best}(l)} r(f)}{|F^b_{best}(l)| + 1}, \frac{C_{best}(l)}{|F_{best}(l)| + 1}\}$$

If a new flow $f_0$ is routed through $l$ and $l$ is the bottleneck link of $f_0$, $r(l)$ is an approximation of the bandwidth allocated to every flow in $F^b_{best}(l) \bigcup \{f_0\}$.

At every node in the network, the link state algorithm collects all $r(l), l \in E$. When a new flow $f_0$ arrives, the source routing is done by using the Bellman-Ford algorithm to find a path $p$ which maximizes the smallest $r(l)$ on the path, i.e. to maximize $\min_{l \in p}\{r(l)\}$. If there are multiple such paths, choose one which maximizes the second smallest $r(l)$, and so on. This procedure continues until a single path is found or a pre-determined maximum allowable length has been reached.

## 4.4 Dynamic Bandwidth Allocation

The bandwidth allocated to a flow changes as other flows join and leave the network dynamically. The source of a flow must adjust its data rate according to the network dynamics. The *dynamic bandwidth allocation* is used to adjust on the fly the bandwidth of all flows and inform the sources to change their data rates accordingly. The design requirement of dynamic bandwidth allocation is that, given any initial state, it must be able to converge to the maxmin allocation in finite time if there is no further arrival of new flows and no further leave of existing flows. Anna Charny et al [5] proposed a distributed algorithm which fulfills such a requirement. We modify the algorithm in order to maintain $F^b_{best}(l)$ and $r(l)$ at each link $l$ and therefore provide the information needed by the approximation algorithm in Section 4.3. The value $r(l)$ will be sent to every router in the network by the link-state algorithm for the purpose of maxmin routing.

1. The source of each flow $f$ sends out forward control messages, RM cells in the ATM context, along its route periodically to determine the expected maxmin bandwidth. The rate of control messages should be bounded by certain low percentage of the average data rate.

2. When a link receives a forward control message, it assigns bandwidth $\frac{C_{best}(l) - \sum_{f \in F_{best} - F^b_{best}(l)} r(f)}{|F^b_{best}(l)|}$ to the message [8] and forwards the message to the

[7]Note that the monitoring as well as other link operations is in fact done by the node in charge of the link.

[8]In the above formula, we ignore the bandwidth consumed by the control messages for the purpose of simplicity. Readers are referred to [5], where the traffic volume of control messages are considered.

next hop on its route. By traversing every link on its route, the forward control message keeps the smallest assigned bandwidth, and the link which assigns such a bandwidth is also kept as the bottleneck link.

3. When the destination receives a forward control message, it turns the message around as a backward control message which traverses back along the route to the source.

4. When a link receives a backward control messages, it checks whether it is the bottleneck link for this flow. If it is, $F^b_{best}(l) = F^b_{best}(l) + \{f\}$; otherwise, $F^b_{best}(l) = F^b_{best}(l) - \{f\}$. $r(l)$ is recomputed when necessary.

5. When the source receives a backward control message, it adjusts the data rate according to the smallest assigned bandwidth carried back by the message. [9]

## 5 Integrated Packet Scheduling

We propose an integrated hierarchical packet scheduling scheme which provides the network support for both QoS and best-effort flows.

### 5.1 Design Goals

When there exist many concurrent flows in the network, it is crucial that the limited bandwidth and other resources are shared effectively and fairly among all competing flows. We have two design goals for the scheduling of data packets:

1. **Guaranteeing QoS:** The bandwidth requirement $B(f)$ must be guaranteed for each $f \in F_{qos}$. The total bandwidth needed by the QoS flows on link $l$ is $C_{qos}(l) = \sum_{f \in F_{qos}(l)} B(f)$. The QoS guarantee is provided in a use-or-lose-it sense: A bandwidth of $B(f)$ is guaranteed for $f$ ($\in F_{qos}(l)$) as long as the incoming data rate of $f$ is high enough to consume it; however, the reserved bandwidth for the current moment, if not used, will not be added up to the bandwidth entitled to be used for the next moment.

2. **Ensuring fairness**: After the bandwidth for the QoS flows is taken off, the rest of the bandwidth, $C_{best}(l) = C(l) - \sum_{f \in F_{qos}(l)} B(f)$, is *shared equally* among all flows in $F_{best}(l)$. Each best-effort flow is entitled to the same share of bandwidth as any other best-effort flow on the link. A flow may receive a less share due to insufficient input data because of an upstream bottleneck link.

[9] If the assigned bandwidth is much greater than the current data rate, the source may choose to increase the data rate gradually in order to avoid oscillation or temporary overloading of the bottleneck link.

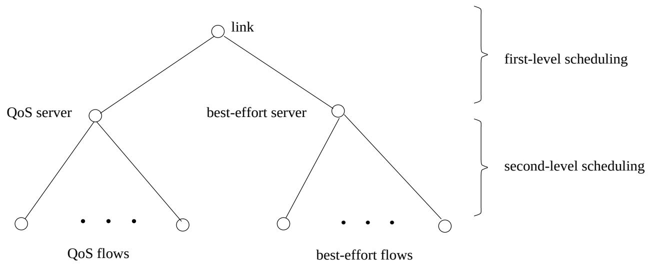

Figure 2: two-level hierarchical scheduling

### 5.2 Hierarchical Scheduling

We propose a hierarchical scheduling algorithm which achieves the two design goals. Our algorithm is similar to the one proposed by Bennett and Zhang [2]. However, it is tuned toward the specific needs raised by the co-existence of QoS flows and best-effort flows.

#### 5.2.1 Overview

A packet scheduling algorithm *operates on each individual link $l$*. The algorithm is a two-level hierarchy as shown in Figure 2. On the first level, the link capacity is divided between two logical scheduling servers: the *QoS server* and the *best-effort server*. The capacity of the QoS server is $C_{qos}(l) = \sum_{f \in F_{qos}(l)} B(f)$, and the capacity of the best-effort server is $C_{best}(l) = C(l) - \sum_{f \in F_{qos}(l)} B(f)$. The values of $C_{qos}(l)$ and $C_{best}(l)$ change when the flow set $F_{qos}(l)$ changes. On the second level, the QoS server schedules the flows in $F_{qos}(l)$ and the best-effort server schedules the flows in $F_{best}(l)$. The QoS server guarantees that every flow $f$ in $F_{qos}(l)$ receives a bandwidth of $B(f)$. The best-effort server makes sure that every flow in $F_{best}$ receives an equal share of $C_{best}(l)$ according to maxmin bandwidth allocation.

### 5.2.2 QoS Server

The QoS server must maintain two invariants.

I1. The capacity of the QoS server, $C_{qos}(l)$, must be $\sum_{f \in F_{qos}(l)} B(f)$ at any time. Whenever a new QoS flow $f$ joins in $F_{qos}(l)$, $C_{qos}(l)$ must be increased by $B(f)$ immediately; whenever an existing QoS flow $f$ leaves $F_{qos}(l)$, $C_{qos}(l)$ must be decreased by $B(f)$.

I2. $\forall f \in F_{qos}(l)$, the QoS server assigns a bandwidth no less than $B(f)$ to $f$, regardless the dynamics of the network state.

### 5.2.3 Best-effort Server

The best-effort server has two properties.

P1. The capacity of the best-effort server, $C_{best}(l)$, is always equal to the link bandwidth left over by the QoS server. When a new QoS flow joins and thus $C_{qos}(l)$ increases, $C_{best}(l)$ must decrease accordingly; when an existing QoS flow leaves and thus $C_{qos}(l)$ decreases, $C_{best}(l)$ must increase accordingly[10]

P2. The best-effort server distributes its capacity $C_{best}(l)$ fairly among all flows in $F_{best}(l)$. Any two flows whose packet queues remain back-logged should receive the same share of bandwidth. The flows whose queues are not back-logged receive less bandwidth which is equal to the incoming data rate.

## 5.3 Implementation

The implementation of the hierarchical scheduling algorithm consists of three parts: (1) scheduling within the QoS server, (2) scheduling within the best-effort server and (3) scheduling between the two servers.

```
// two-level hierarchical scheduling on link $l$
while true do
(1)   a packet $n$ is selected from flows in $F_{qos}(l)$
      by the QoS server
(2)   a packet $m$ is selected from flows in $F_{best}(l)$
      by the best-effort server
(3)   select $n$ or $m$ for transmission by top scheduler
```

[10]Note that $C_{qos}(l)$ has an upper bound ($\lambda C(l)$, where $\lambda < 1$ and $C(l)$ is the overall link capacity) as specified in Section 2.

Referring to Figure 2, (1) and (2) are the second-level schedulings; (3) is the first-level scheduling. All three parts can be implemented by the weighted fair queuing [8].

### 5.3.1 Scheduling within QoS Server

Assume the invariant I1 (Section 5.2.2) always holds, i.e., the capacity of the QoS server, $C_{qos}(l)$, is always $\sum_{f \in F_{qos}(l)} B(f)$. How to maintain such a capacity in a dynamic network is discussed in Section 5.3.3. We implement the scheduling within the QoS server by the weighted fair queuing as follows.

1. A packet queue is maintained for each flow $f \in F_{qos}(l)$. The arrival packets are inserted into the queue in the FIFO order. A timestamp $t^i_{qos}$ is calculated for the $i$th arrival packet.

   $$t^i_{qos} \leftarrow max\{V_{qos}, t^{i-1}_{qos}\} + \frac{sp_i}{B(f)}$$

   where $V_{qos}$ is the reference *virtual time* [2] of the QoS server, $sp_i$ is the length of the packet, $t^{i-1}_{qos}$ is the timestamp of the $(i-1)$th packet and $B(f)$ is used as the *weight*. See Table 3 for a quick reference to the notations introduced here and in the following paragraphs.

   $V_{qos}$ is a variable maintained by the QoS server, keeping track of the timestamp of the last transmitted packet from $F_{qos}(l)$. It is used to determine where the timestamp of a new or resumed QoS flow should start. Note that there is a single variable $V_{qos}$ used by all flows in $F_{qos}(l)$.

2. The scheduling among flows in $F_{qos}(l)$ is based on the timestamps. Whenever the QoS server becomes idle, the packet with the smallest timestamp among all queues is selected for transmission.

The above weighted fair queuing assigns bandwidth to flows based on their weights. The bandwidth received by $f \in F_{qos}(l)$ is equal to $\frac{B(f)}{\sum_{f' \in F_{qos}(l)} B(f')} \times C_{qos}(l) = \frac{B(f)}{C_{qos}(l)} \times C_{qos}(l) = B(f)$, if all flows are back-logged. Hence, the invariant I2 holds. Readers are referred to [2, 8, 21] for the detailed study of fair queuing.

| | |
|---|---|
| $a_i$ | the arrival time of the $i$th data packet |
| $sp_i$ | the size of the $i$th packet in term of number of bits |
| $t^i_{qos}$ | the timestamp assigned to the $i$th packet by the QoS server for scheduling within the server |
| $t^i_{best}$ | the timestamp assigned to the $i$th packet by the best-effort server for scheduling within the server |
| $T^i_{qos}$ | the timestamp assigned to the $i$th packet by the QoS server for scheduling between the QoS and the best-effort servers |
| $T^i_{best}$ | the timestamp assigned to the $i$th packet by the best-effort server for scheduling between the QoS and the best-effort servers |
| $V_{qos}$ | the reference *virtual time* of the QoS server, keeping track of the timestamp of the last transmitted packet from $F_{qos}(l)$ |
| $V_{best}$ | the reference *virtual time* of the best-effort server, keeping track of the timestamp of the last transmitted packet from $F_{best}(l)$ |

Table 3: Additional notations for the hierarchical scheduling

### 5.3.2 Scheduling within Best-effort Server

Assume the property P1 (Section 5.2.3) always holds, i.e., the capacity of the best-effort server, $C_{best}(l)$, is always equal to $C(l) - \sum_{f \in F_{qos}(l)} B(f)$. How to achieve this will be discussed shortly. We implement the scheduling within the best-effort server by the weighted fair queuing as follows. The property P2 is achieved by assigning an equal weight to every flow.

1. A packet queue is maintained for each flow $f \in F_{best}(l)$. The arrival packets are inserted into the queue in the FIFO order. The weight of each flow is 1. A timestamp $t^i_{best}$ is calculated for the $i$th arrival packet of $f$.

$$t^i_{best} \leftarrow max\{V_{best}, t^{i-1}_{best}\} + \frac{sp_i}{C_{best}(l)/|F_{best}(l)|}$$

   $V_{best}$ is a variable maintained by the best-effort server, keeping track of the timestamp of the last transmitted packet from $F_{best}(l)$. $V_{best}$ is used as a reference *virtual time* of the server to determine where the timestamp of a new or resumed best-effort flow should start.

2. The scheduling among flows in $F_{best}(l)$ is based on the timestamps. Whenever the best-effort server becomes idle, the packet with the smallest timestamp among all non-empty queues is selected for transmission.

We have three observations about the above scheduling.

O1. Those flows whose queues remain back-logged receive the same share of bandwidth from the best-effort server because they have the same weight of 1.

O2. There may exist flows with empty queues due to insufficient incoming data packet rate, which may result from an upstream bottleneck link. These flows consume less bandwidth than the others simply because at times there are no packets in the queues for scheduling. Because the queue is not back-logged, the outgoing data rate, which is the actual bandwidth consumed, must be equal to the incoming data rate.

O3. Our scheduling is work-conserving, which means the capacity of the best-effort server will be fully utilized as long as there are back-logged queues.

Additional flexibility may be achieved by assigning different weights to different types of flows. Some interactive flows demand relatively small bandwidth. However, the instant bandwidth availability is critical to their performance. Examples are distributed games such as playing chess or cards over the Internet. Some other flows are relatively bandwidth-insensitive. Examples are non-interactive video retrieval and large file transmission working in the background. We can modify the scheduling of the best-effort server by classifying the flows into different categories, to each of which a different weight $w$ is assigned. The timestamp calculation becomes $t^i_{best} \leftarrow max\{V_{best}, t^{i-1}_{best}\} + \frac{sp_i}{w}$. The flows with larger weights receive more prompt service and/or larger bandwidth shares. For the most critical flows, a special timestamp of $-1$ is assigned to every of their packets so that the packets will always be transmitted before those of other flows.

### 5.3.3 Scheduling between Two Servers

The QoS server and the best-effort server are *logical servers* using the same physical link. When both servers have packets to send, we must select one of them for the actual transmission. We want the scheduling between the two servers satisfies the invariant I1 (Section 5.2.2) and the property P1 (Section 5.2.3), i.e., the QoS server receives a capacity of $\sum_{f \in F_{qos}(l)} B(f)$ and the best-effort server receives a capacity of $C(l) - \sum_{f \in F_{qos}(l)} B(f)$.

The weighted fair queuing is used again, where the two servers are modeled as two logical flows, whose packets are from the physical flows in $F_{qos}(l)$ ($F_{best}(l)$) sorted by the timestamps. Let the weight of the QoS server be $W_{qos} = \sum_{f \in F_{qos}(l)} B(f)$ and that of the best-effort server be $W_{best} = C(l) - \sum_{f \in F_{qos}(l)} B(f)$. $W_{qos}$ and $W_{best}$ are not fixed in the run-time; they change when $F_{qos}(l)$ changes.

1. The $i$th packet selected by the QoS server is assigned a timestamp

   $$T^i_{qos} \leftarrow max\{V_{link}, T^{i-1}_{qos}\} + \frac{sp_i}{W_{qos}}$$

   where $T^{i-1}_{qos}$ is the timestamp assigned to the $(i-1)$th packet selected by the QoS server. $V_{link}$ will be explained shortly. The $i$th packet selected by the best-effort server is assigned a timestamp

   $$T^i_{best} \leftarrow max\{V_{link}, T^{i-1}_{best}\} + \frac{sp_i}{W_{best}}$$

   where $T^{i-1}_{best}$ is the timestamp assigned to the $(i-1)$th packet selected by the best-effort server.

   $V_{link}$ is a variable maintained by the physical link,[11] keeping track of the timestamp — $T^i_{qos}$ or $T^i_{best}$ depending which server the packet is from — of the last packet transmitted by the physical link. $V_{link}$ is used as a reference *virtual time* of the link to determine where the timestamp should start when a packet arrives at an empty QoS or best-effort server,

2. When both servers select packets, the packet with the smaller timestamp will be transmitted.

The bandwidth received by the QoS server is $\frac{W_{qos}}{W_{qos}+W_{best}} \times C(l) = \frac{W_{qos}}{C(l)} \times C(l) = W_{qos} = \sum_{f \in F_{qos}(l)} B(f)$, and the bandwidth received by the best-effort server is $\frac{W_{best}}{W_{qos}+W_{best}} \times C(l) = W_{best} = C(l) - \sum_{f \in F_{qos}(l)} B(f)$.

[11] In more precise words, by the node in charge of the link.

### 5.3.4 Overhead

We study the per-packet computational overhead of our algorithm. For scheduling within the QoS server, finding the smallest timestamp among all flows in $F_{qos}(l)$ takes $O(log|F_{qos}(l)|)$, if a balanced binary tree such as a heap tree is maintained. For scheduling within the best-effort server, finding the smallest timestamp takes $O(log|F_{best}(l)|)$. For scheduling between the QoS server and the best-effort server, finding the smaller timestamp takes $O(1)$. Two timestamps, $t^i_{qos}$ and $T^i_{qos}$ or $t^i_{best}$ and $T^i_{best}$, are calculated for each packet, which takes a small constant time. Therefore, the total overhead for scheduling a single packet is $O(log|F_{qos}(l)| + log|F_{best}(l)|)$, which is reasonably small and comparable to the time complexity $O(log|F(l)|)$ of the single-level fair queuing scheduling.

## 6 Conclusion

We presented several possible algorithms for routing and scheduling which allow coexistence of QoS and best effort flows in future high-speed networks. Our network routing algorithms took into account state imprecision in routers, maxmin bandwidth allocation, and existing link state information. Our scheduling algorithms enforced effective and guaranteed bandwidth allocation for QoS flows, and fair sharing of bandwidth for best effort flows.

In summary, our integrated routing and scheduling framework allows for

- bandwidth QoS routing when intermediate nodes carry imprecise state information which is a realistic assumption in current and future networks.

- finding a best-effort route according to fairness-throughput performance relation. This type of relation optimizes the maxmin bandwidth allocation, hence with our maxmin routing algorithm we find a route which will be optimized according to the maxmin bandwidth allocation.

- approximation algorithm to find best-effort routes according to maxmin bandwidth allocation using link states only.

- starvation avoidance in case of best effort flows because we maintain an upper bound on the bandwidth allocation for QoS flows, which never encompasses the entire link bandwidth.
- maximal throughput/link utilization because we allow for sharing of bandwidth by best-effort flows which is not utilized by QoS flows. This means that if only few QoS flows are routed and scheduled through a link, the remaining bandwidth can be shared among best-effort flows.

Our future work will consider other possible QoS routing algorithms (distributed, hierarchical) for multiple QoS requirements and their coexistence with best-effort routing algorithms. Within the two-level hierarchical scheduling, we will investigate and analyze adaptive mechanisms to incorporate scheduling of QoS flows with range parameter specification $(QoS_{min}, QoS_{max})$ and their impact on best-effort scheduling.

## References

[1] B. Awerbuch and Y. Shavitt. Converging to approximated max-min flow fairness in logarithmic time. *accepted for Infocom'98, San-Francisco, CA*, April 1998.

[2] J. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *ACM SIGCOMM'96*, 1996.

[3] D. Bertsekas and R. Gallager. Data networks. *Prentice Hall, Englwood Cliffs, N.J.*, 1992.

[4] F. Bonomi and K. Fendick. The rate-based flow control framework for the available bit rate atm service. *IEEE Network*, pages 9(2):25–39, March/April 1995.

[5] A. Charny, D. D. Clark, and R. Jain. Congestion control with explicit rate indication. *IEEE International Conference on Communications*, 1995.

[6] S. Chen and K. Nahrstedt. Maxmin fair routing in connection-oriented networks. *Tech. Rep., Dept. of Com. Sci., UIUC, USA*, 1998.

[7] S. Chen and K. Nahrstedt. On finding multi-constrained paths. *IEEE International Conference on Communications*, June 1998.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM'89*, pages 3–12, 1989.

[9] J. M. Faffe. Bottleneck flow control. *IEEE Transactions on Communications*, COM-29(7):954–962, July 1981.

[10] ATM Forum. Private network network interface (pnni) v1.0 specifications. May 1996.

[11] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman and Co., 1979.

[12] R. Guerin and A. Orda. Qos-based routing in networks with inaccurate information: Theory and algorithms. *Infocom'97, Japan*, April 1997.

[13] H. T. Kung, T. Blackwell, and A. Chapman. Credit-based flow control for atm networks: Credit update protocol, adaptive credit allocation, and statistical multiplexing. *ACM SIGCOMM'94*, pages 101–114, August 1994.

[14] D. H. Lorenz and A. Orda. Qos routing in networks with uncertain parameters. *Infocom'98*, March 1998.

[15] Q. Ma and P. Steenkiste. Quality-of-service routing with performance guarantees. *Proceedings of the 4th International IFIP Workshop on Quality of Service*, May 1997.

[16] Q. Ma, P. Steenkiste, and H. Zhang. Routing high-bandwidth traffic in max-min fair share networks. *Proceedings of SIGCOMM'96*, August 1996.

[17] J. Mosley. Asynchronous distributed flow control algorithms. *Ph.D. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA*, 1984.

[18] J. Moy. Ospf version 2, internet rfc 1583. March 1994.

[19] C. Parris, H. Zhang, and D. Ferrari. Dynamic management of guaranteed performance multimedia connections. *Multimedia Systems Journal*, 1:267–283, 1994.

[20] Z. Wang and J. Crowcroft. Qos routing for supporting resource reservation. *JSAC*, September 1996.

[21] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. *ACM SIGCOMM'91*, 1991.