Hierarchical Scheduling for Multiple Classes of Applications in Connection-Oriented Integrated-Service Networks *

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

Abstract

Multiple classes of conventional and multimedia applications are expected to be supported by the emerging connection-oriented integrated-service networks, where best-effort flows and quality-of-service (QoS) flows co-exist. We propose a novel two-level hierarchical scheduling algorithm, which provides an integrated, simple scheme to distribute the link bandwidth among a dynamic set of flows in the network, such that (1) every QoS flow always receives a bandwidth which guarantees the required quality of service and (2) after the requirements of the QoS flows are satisfied, the rest network bandwidth is fairly shared among all best-effort and soft-QoS flows. We extend our hierarchical scheduling algorithm to support both bursty and non-bursty flows. We also discuss the techniques which make the throughput of a QoS flow to be constantly lower bounded and/or upper bounded.

1 Introduction

In traditional connectionless networks, data packets of a flow (session) may follow different routes to the destination node. This architecture does not meet the requirements of the future integrated-service networks. It does not support resource reservation which is vital for the provision of guaranteed quality of service (QoS). Hence, the next generation of high-speed wide-area networks is likely to be connection-oriented so that the data packets of a flow are transmitted along the same route in the FIFO order. Extensive research and experiments have been done with the connection-oriented ATM technology [2, 3].

Multiple classes of applications are expected to be supported by the integrated-service networks. While the conventional applications such as *e-mail* and *ftp* send textual or binary data using *best-effort flows*, the emerging distributed multimedia applications require real-time data to be delivered through *quality-of-service (QoS) flows*, which demand certain system resources to be reserved in order to ensure the acceptable quality. The QoS flows have very diverse traffic patterns. First, some flows such as MPEG video are bursty, while some others such as audio require constant throughput. Second, some flows such as mission-critical real-time control streams require the throughput to be *constantly* greater than a minimum rate, while some others require the throughput to be both lower-bounded and upper-bounded. A too-large throughput may not be acceptable due to the limited processing capacity at the receiver end. Third, some flows are *adaptable* in the sense that the sending rates can be adjusted according to the current bandwidth availability, while some others are *non-adaptable* in the sense that they can only operate appropriately at certain rates. Different types of flows impose different requirements on the system support. Though many scheduling policies have been proposed for each of the above flows [6, 7, 10], none of the policies provides a simple solution to all of them.

Bennett and Zhang proposed a nice hierarchical scheduling framework [1]. We extend the idea of hierarchical scheduling and tailor it towards the specific needs raised by a set of well-defined application classes. Our hierarchical scheduling algorithm can support both best-effort and QoS flows, both bursty and non-bursty flows, both throughput-lower-bounded and throughput-upper-bounded flows, and both adaptable and non-adaptable flows.

2 Network Model and Flow Characteristics

A network is modeled as a graph $\langle N, E\rangle$, where N is a set of nodes which are fully connected by a set E of full-duplex, directed communication links. Each link l has a bandwidth capacity $C(l)$. Let F be the set of flows in the network. We study the connected-oriented network 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 [4]. 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 $F(l)$.

Each flow f is characterized by a pair of bandwidth requirements $\langle B_{min}(f), B_{max}(f)\rangle$ [8], where $B_{min}(f)$ is the minimum bandwidth that ensures an acceptable quality, and $B_{max}(f)$ is the maximum (or peak) bandwidth that is constrained by the highest source rate. Flows can be classified into the following categories.

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

0-7695-0253-9/99 $10.00 © 1999 IEEE

Best-effort flow: If $B_{min}(f) = 0$, f is called a *best-effort* flow. Examples are file transmission (*ftp*), web-page download, and database retrieval.

QoS flow: If $B_{min}(f) \neq 0$, f is called a *QoS flow.*

1) **Soft-QoS flow:** If $B_{min}(f) < B_{max}(f)$, f is called a *soft-QoS flow.* The minimum data throughput is given by $B_{min}(f)$ and must be guaranteed by the network. However, the actual data rate can be anywhere between $B_{min}(f)$ and $B_{max}(f)$, and is subject to the dynamics of bandwidth availability in the network. In ATM networks, the available bit rate (ABR) service is an example.

2) **Hard-QoS flow:** If $B_{min}(f) = B_{max}(f)$, f is called a *hard-QoS flow.* The data throughput is required to be a constant. In ATM networks, the constant bit rate (CBR) service is an example.

The variable bit rate (VBR) service in ATM networks can also be modeled by QoS flows. However, additional specification about the burstiness of the flow such as the maximum burst duration (burst tolerance) must be provided.

3 Integrated Packet Scheduling

We propose a new integrated hierarchical packet scheduling scheme which provides the network support for multiple classes of conventional and multimedia applications with various flow types including best-effort, soft-QoS and hard-QoS flows.

3.1 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 primary design goals for bandwidth allocation.

Guaranteeing QoS: The minimum bandwidth requirement $B_{min}(f)$ must be guaranteed for each $f \in F$.

Ensuring fairness: After the bandwidth for guaranteeing the minimum QoS is taken off, the rest bandwidth, $C(l) - \sum_{f \in F(l)} B_{min}(f)$, is *shared equally* among all flows in $F(l)$.

There are many other design issues besides the above two, such as handling burstness and ensuring a strict upper bound $B_{max}(f)$ in order to prevent buffer overflow. All the above issues will be discussed in the rest of this section.

3.2 Hierarchical scheduling

We propose a decentralized hierarchical scheduling scheme which achieves the two design goals.

A packet scheduling algorithm *operates on each individual link l.* The algorithm is a two-level hierarchy as shown in Figure 1. At 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 $C_{qos}(l) = \sum_{f \in F(l)} B_{min}(f)$, and the capacity of the best-effort server is $C_{best}(l) = C(l) - \sum_{f \in F(l)} B_{min}(f)$. Note the values of

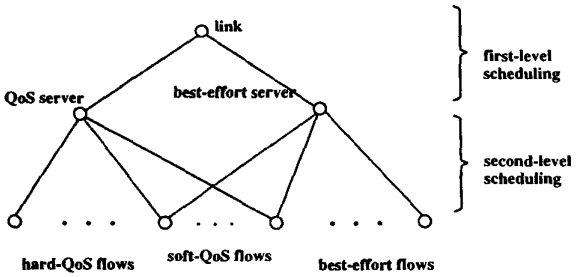

Figure 1. The QoS server schedules the QoS flows and the best-effort server schedules the soft-QoS and best-effort flows.

$C_{qos}(l)$ and $C_{best}(l)$ change when the flow set $F(l)$ changes, which, as shown later, is critical for guaranteeing a minimum required bandwidth for every QoS flow seamlessly in a dynamic network. At the second level, the QoS server ensures that each flow f receives a fixed bandwidth of $B_{min}(f)$, and the best-effort server assigns each flow an additional share of bandwidth which is not fixed but subject to the dynamics of $C_{best}(l)$.

The QoS server only schedules the QoS flows because, for any best-effort flow, $B_{min}(f) = 0$. The set of QoS flows is denoted as $F_{qos}(l)$. On the other hand, the best-effort server does not have to schedule the hard-QoS flows because $B_{min}(f) = B_{max}(f)$ and the QoS server already provides all bandwidth they need. The set of soft-QoS and best-effort flows scheduled by the best-effort server is denoted as $F_{best}(l)$. Note that $F_{qos}(l) \bigcup F_{best}(l) = F(l)$ and $F_{qos}(l) \bigcap F_{best}(l)$ is the set of soft-QoS flows on link l.

QoS server: The QoS server must maintain two invariants.

I1. $C_{qos}(l) = \sum_{f \in F_{qos}(l)} B_{min}(f)$. Whenever a new QoS flow f joins in $F_{qos}(l)$, $C_{qos}(l)$ must be increased by $B_{min}(f)$ immediately; whenever an existing QoS flow f leaves $F_{qos}(l)$, $C_{qos}(l)$ must be decreased by $B_{min}(f)$.

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

Best-effort server: The best-effort server has two properties.

P1. $C_{best}(l) = C(l) - C_{qos}(l)$. The capacity of the best-effort server 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.

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 backlogged should receive the

same share of bandwidth. The flows whose queues are empty receive less bandwidth which is equal to the average incoming data rate.

3.3 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 QoS server
(2) a packet m is selected from flows in $F_{best}(l)$ by best-effort server
(3) select one from n and m for transmission

Referring to Figure 1, (1) and (2) are the second-level schedulings; (3) is the first-level scheduling. All three parts can be implemented by the weighted fair queueing [5].

3.3.1 Scheduling within the QoS server

Assume the invariant I1 (Section 3.2) always holds, i.e., the capacity of the QoS server, $C_{qos}(l)$, is always $\sum_{f \in F_{qos}(l)} B_{min}(f)$. How to maintain such a capacity in a dynamic network is discussed in Section 3.3.3.

In order not to starve the best-effort server, we also assume $C_{qos}(l) \leq \alpha \times C(l)$, where α (< 1) is a system constant specifying the maximum percentage of the link capacity used by the QoS server. This can be achieved by the admission control.

We implement the scheduling within the QoS server by the weighted fair queueing as follows. 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 $t^i_{qos}(f)$ is calculated for the ith arrival packet.

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

where V_{qos} is the reference *virtual time* [1] of the QoS server, $p_i(f)$ is the length of the ith packet, $t^{i-1}_{qos}(f)$ is the timestamp of the $(i-1)$th packet and $B_{min}(f)$ is used as the *weight*. 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 $F_{qos}(l)$. The timestamp t^i_{qos} specifies the *expected transmission completion time* of the ith packet [11].

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 queueing assigns bandwidth to flows based on their weights. The bandwidth received by $f \in F_{qos}(l)$ is equal to $\frac{B_{min}(f)}{\sum_{f' \in F_{qos}(l)} B_{min}(f')} \times C_{qos}(l) = \frac{B_{min}(f)}{C_{qos}(l)} \times C_{qos}(l) = B_{min}(f)$, if all flows are backlogged. Hence, the invariant I2 holds. Readers are referred to [1, 5] for the detailed study of fair queueing.

3.3.2 Scheduling within the best-effort server

Assume the property P1 (Section 3.2) always holds, i.e., the capacity of the best-effort server, $C_{best}(l)$, is always equal to $C(l) - \sum_{f \in F_{qos}(l)} B_{min}(f)$. How to achieve this will be discussed shortly.

We implement the scheduling within the best-effort server by the weighted fair queueing as follows. 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 $t^i_{best}(f)$ is calculated for the ith arrival packet of f.

$$t^i_{best}(f) \leftarrow max\{V_{best}, t^{i-1}_{best}(f)\} + p_i(f)$$

V_{best} is a variable maintained by the best-effort server, keeping track of the timestamp of the last transmitted packet from $F_{best}(l)$.

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.

The property P2 is achieved by assigning an equal weight to every flow. The flows whose queues remain backlogged receive the same share of bandwidth from the best-effort server because they have the same weight of 1.

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. The sporadic and bursty nature of their traffic makes it undesired to reserve a fixed portion of bandwidth on the QoS server. Some other flows are relatively bandwidth-insensitive. Examples are non-interactive video retrieval and large file transmission working on 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 $t^i_{best}(f) \leftarrow max\{V_{best}, t^{i-1}_{best}(f)\} + \frac{p_i(f)}{w}$. 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.

3.3.3 Scheduling between the 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 and the property P1 (Section 3.2).

The weighted fair queueing 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 $W_{qos} = \sum_{f \in F_{qos}(l)} B_{min}(f)$ and that of the best-effort server be $W_{best} = C(l) - \sum_{f \in F_{qos}(l)} B_{min}(f)$. W_{qos} and W_{best} are not fixed in the run-time; they change when $F_{qos}(l)$ changes.

The ith packet selected by the QoS server is assigned a timestamp

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

where p_i is the size of the packet and 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 ith packet selected by the best-effort server is assigned a timestamp

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

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,[1] 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.

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

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

3.3.4 Scheduling interference

The above hierarchical scheduling works fine if $F_{qos}(l)$ and $F_{best}(l)$ are disjointed, i.e. there are no soft-QoS flows. However, if $F_{qos}(l) \bigcap F_{best}(l) \neq \emptyset$, a scheduling interference problem arises.

Suppose $f_1 \in F_{qos}(l) \bigcap F_{best}(l)$. Though f_1 is scheduled by both QoS server and best-effort server, it has a single packet queue as every other flow does. The QoS server assigns f_1 a bandwidth of $B_{min}(f_1)$. Suppose the best-effort server assigns f_1 a bandwidth of $B_{best}(f_1)$. We want the total bandwidth of f_1 to be $B_{min}(f_1) + B_{best}(f_1)$. However, due to the scheduling interference between the two servers, the actual bandwidth received by f_1 is $max\{B_{min}(f_1), B_{best}(f_1)\}$, which is explained by the following example in Figure 2.

Let $B_{min}(f_1) = 1$ and $B_{max}(f_1) = +\infty$. Suppose f_2 is a best-effort flow with $B_{min}(f_2) = 0$ and $B_{max}(f_2) = +\infty$. f_1 is scheduled by both servers, while f_2 is scheduled only by the best-effort server. Suppose all packets arrive at time 0 and the length of each packet is 1. The ith packet of f_1 (f_2) is denoted as p^i_1 (p^i_2). The timestamps of all packets are shown in the figure. Suppose the link capacity is 2. $C_{qos}(l) = B_{min}(f_1) = 1$ since f_1 is the only flow in $F_{qos}(l)$, and thus $C_{best}(l) = 2 - 1 = 1$. Ideally, the QoS server assigns bandwidth 1 to f_1 and the best-effort server assigns bandwidth 0.5 to both f_1 and f_2 so that the total bandwidth of f_1 is three times that of f_2.

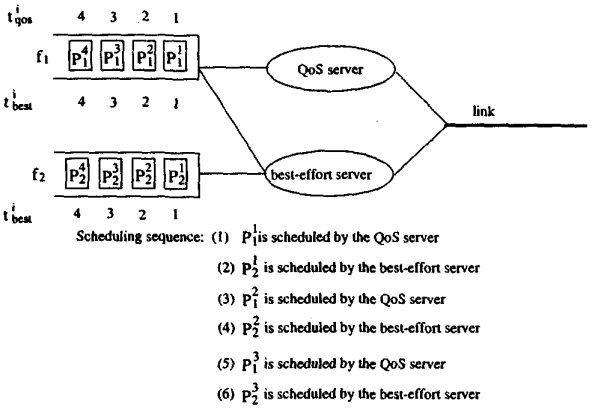

Figure 2. $B_{min}(f_1) = 1$, $B_{max}(f_1) = +\infty$, $B_{min}(f_2) = 0$, and $B_{max}(f_2) = +\infty$. Ideally, $B(f_1) = 1.5$ and $B(f_2) = 0.5$, where $B(f_1)$ and $B(f_1)$ are the bandwidth received by f_1 and f_2, respectively. However, due to scheduling interference, $B(f_1) = B(f_2) = 1$.

However, let us show how the scheduling interference occurs. Since the capacities of the two servers are the same, we assume they occupy the link alternatively. First, the QoS server schedules and transmits p^1_1. After that, the first packet in the f_1 queue is p^2_1 with $t^2_{best}(f_1) = 2$. Second, the best-effort server schedules p^1_2 which has the smallest timestamp $t^1_{best}(f_2) = 1$. Third, the QoS server transmits p^2_1. Fourth, the best-effort server transmits p^2_2 ... The packet transmission sequence is therefore, p^1_1, p^1_2, p^2_1, p^2_2, p^3_1, p^3_2, and so on. The packets in f_1 and f_2 are transmitted alternatively. f_1 and f_2 receive an equal bandwidth of 1.

The reason is that, after p^1_1 is scheduled by the QoS server, the best-effort timestamp of the next packet, p^2_1, does not reflect the fact that p^1_1 is not scheduled by the best-effort server and thus does not consume the bandwidth of the best-effort server. The scheduling priority of f_1 by the best-effort server, which is solely determined by the timestamp, *should* remain the same (but *actually* decreases) after p^1_1 is scheduled by the QoS server. A simple solution is to decrease the best-effort timestamp of p^2_1 so that $t^2_{best}(f_1) = 1$ after p^1_1 is sent. A more general and efficient solution is described next.

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

3.3.5 Solution to scheduling interference

The problem of scheduling interference can be solved by modifying the way the timestamps are calculated. In the following, we only consider the soft-QoS flows. The hard-QoS and best-effort flows do not have the interference problem, and the calculation of their timestamps remains the same as discussed in Sections 3.3.1-3.3.2.

Without losing generality, consider the ith packet of a soft-QoS flow f. The timestamps, $t^i_{qos}(f)$ and/or $t^i_{best}(f)$, are computed only when the packet becomes the first one in the queue. Two additional variables, $t^{pre}_{qos}(f)$ and $t^{pre}_{best}(f)$, are maintained for each soft-QoS flow to keep track of the timestamps of the last transmitted packet. Initially, $t^{pre}_{qos}(f) = V_{qos}$ and $t^{pre}_{best}(f) = V_{best}$ when the flow starts.

1. If the ith packet arrives at an empty queue,

$$t^i_{qos}(f) \leftarrow max\{V_{qos}, t^{pre}_{qos}(f)\} + \frac{p_i(f)}{B_{min}(f)}$$
$$= V_{qos} + \frac{p_i(f)}{B_{min}(f)}$$

$$t^i_{best}(f) \leftarrow max\{V_{best}, t^{pre}_{best}(f)\} + p_i(f) = V_{best} + p_i(f)$$

By the definition, $V_{qos} \geq t^{pre}_{qos}(f)$ and $V_{best} \geq t^{pre}_{best}(f)$.

2. If the ith packet arrives at a non-empty queue, the calculation of the timestamps is delayed until after the $(i-1)$th packet is transmitted.

If the $(i-1)$th packet is scheduled by the QoS server,

$$t^{pre}_{qos}(f) \leftarrow t^{i-1}_{qos}(f)$$

$$t^i_{qos}(f) \leftarrow t^{pre}_{qos}(f) + \frac{p_i(f)}{B_{min}(f)}$$

$$t^i_{best}(f) \leftarrow t^{pre}_{best}(f) + p_i(f)$$

If the $(i-1)$th packet is scheduled by the best-effort server,

$$t^{pre}_{best}(f) \leftarrow t^{i-1}_{best}(f)$$

$$t^i_{qos}(f) \leftarrow t^{pre}_{qos}(f) + \frac{p_i(f)}{B_{min}(f)}$$

$$t^i_{best}(f) \leftarrow t^{pre}_{best}(f) + p_i(f)$$

If the $(i-1)$th packet is scheduled by the QoS server, only $t^{pre}_{qos}(f)$ is increased while $t^{pre}_{best}(f)$ remains the same so that the scheduling activity of the QoS server does not change the scheduling priority of the best-effort server. If the packet is scheduled by the best-effort server, only $t^{pre}_{best}(f)$ is increased while $t^{pre}_{qos}(f)$ remains the same. By calculating the timestamps in such a way that separates the scheduling activities of the two servers, the scheduling interference is avoided, and the total bandwidth received by f is equal to $B_{min}(f) + B_{best}(f)$, where $B_{best}(f)$ denotes the bandwidth share assigned by the best-effort server.

3.3.6 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 $O(1)$. In the worst case, there are four timestamps, $t^i_{qos}(f)$, $t^i_{best}(f)$, T^i_{qos} and T^i_{best}, to be calculated for a 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 queueing scheduling.

3.3.7 Scheduling of bursty and non-bursty flows

For some bursty QoS flows such as compressed video, $B_{min}(f)$ is given as the *average* bandwidth for guaranteeing the required quality. The *actual* data rate may be higher or lower than $B_{min}(f)$ at times, which is the case of VBR service in ATM networks. The QoS server provides certain degree of tolerance to the burstyness of such flows. When many bursty flows are scheduled by the QoS server, their bursts are likely to interleave. When some flows send packets at their burst rates, some other flows may send packets at their low rates and thus leave bandwidth to absorb the bursts.

We require a bursty flow f to specify the maximum burst duration, $D(f)$. The maximum burst size is $D(f) \times B_{max}(f)$. At link l, each bursty QoS flow f maintains a state variable, called *burst credit* and denoted as $\Omega(f)$. When the flow sends data at a lower rate than $B_{min}(f)$, $\Omega(f)$ is increased; when it sends data at a higher rate than $B_{min}(f)$, $\Omega(f)$ is decreased. The value of $\Omega(f)$ must always be in the range of $[0, D(f) \times B_{max}(f)]$.

In addition to $\Omega(f)$, another variable $\phi(f)$ is maintained to keep the most recent time at which the queue of f becomes empty.

1. Whenever the length of the packet queue of f exceeds a threshold value and $\Omega(f) > 0$, the bursty mode is trigered. $t^i_{qos}(f)$ is calculated based on $B_{max}(f)$ instead of $B_{min}(f)$.

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

$$\Omega(f) \leftarrow max\{\Omega(f) - p_i(f) \times (1 - \frac{B_{min}}{B_{max}}), 0\}$$

2. Whenever $\Omega(f)$ reaches zero, the normal mode is trigered. $t^i_{qos}(f)$ is calculated based on $B_{min}(f)$ as usual.

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

3. Whenever a packet arrives at an empty queue at time t, $\Omega(f)$ is set as follows.

$$\Omega(f) \leftarrow min\{\Omega(f) + (t - \phi(f)) \times B_{min}(f), D(f) \times B_{max}(f)\}$$

Simply speaking, the idea is that the burst credit is built up for the unused bandwidth and then is used when the burst comes. In fact, there are many issues to be addressed in order for this approach to be effective. For example, the maximum burst size $D(f) \times B_{max}(f)$ should not be arbitrarily large. A sufficiently large maximum burst size together with a sufficiently large $B_{max}(f)$ may grab all bandwidth from other flows in a long period.

Another problem is that many (or all in the worst case) flows may enter the bursty phase simultaneously. When the total data rate exceeds the capacity of the QoS server, the backlogs of the queues are built up. If such synchronized bursts persist, the buffer is overflowed and the packets are lost. Besides, a long end-to-end packet delay will be observed. There are a number of approaches to deal with this problem. First, when the length of a queue exceeds certain threshold value, a control message can be sent to the upstream nodes and the source node to slow down the incoming data rate. Second, $B_{min}(f)$ can be temporarily increased for the flows in burst. As a result, the capacity of the QoS server is temporarily increased. Third, the best-effort server can be used to help absorbing the bursts. The soft-QoS flows are already scheduled by the best-effort server. For the hard-QoS flows, we can temporarily insert them into $F_{best}(l)$ until their bursts are transmitted.[2] We can also increase the weights of these flows in the best-effort server so that they can receive a larger bandwidth share until the bursts are over.

In order to prevent the dynamic behavior of the bursty QoS flows from affecting the performance of the non-bursty QoS flows, we can separate the scheduling of bursty flows and non-bursty flows by using different QoS servers. Our proposed hierarchical scheduling algorithm can be easily generalized to handle multiple QoS servers.

Some flows may require their data rates to be always bounded by $B_{max}(f)$. A rate greater than $B_{max}(f)$ is undesired because that may overwhelm the processing ability at the receiver end. We denote the set of such flows as $F_{qos}^{max}(l)$. A simple solution is to make the sending rate at the source to be bounded by $B_{max}(f)$. However, this will not guarantee the incoming rate at the receiver to be always bounded by $B_{max}(f)$ due to buffering and transmission burst at the intermediate links. A better solution is to make the leaky-bucket-like traffic shaping at each link l for each flow f in $F_{qos}^{max}(l)$. Suppose the timestamp of a packet is calculated only when the packet becomes the first one in the queue as discussed in Section 3.3.5. Let the length of the packet be p. Right after the timestamp is calculated, a timer, equal to $\frac{p}{B_{max}(f)}$, is set for the next packet. After the first packet is transmitted, if the timer has expired, the next packet is immediately available for scheduling. Otherwise, it has to wait until the timer expires before its timestamp can be calculated. Hence, the consecutive packets are well spaced by using an appropriately-set timer, which makes the sending rate of f at any intermediate link to be bounded by $B_{max}(l)$.

[2] There are non-bursty hard-QoS flows and hard-QoS flows. The former requires a constant throughput, and the latter satisfies the definition of $B_{min}(f) = B_{max}(f)$ in a statistical sense that the *average* throughput is a constant over periods.

4 Conclusion

We propose a two-level hierarchical scheduling algorithm, which dynamically adjusts the fair-share bandwidth allocated to the flows such that (1) each best-effort flow receives a fair share from the best-effort server, (2) each hard-QoS flow receives the required bandwidth from the QoS server, and (3) each soft-QoS flow receives the minimum required bandwidth from the QoS server and a fair share from the best-effort server. Both servers are implemented by the weighted fair queueing. We discuss the problem of scheduling interference between two servers and provide a solution. We also discuss how to schedule bursty and non-bursty flows by using multiple QoS servers that separate the scheduling activities of different types of flows.

References

[1] J. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. *ACM SIGCOMM'96*, 1996.

[2] S. Berson and L. Berger. IP Integrated Services with RSVP over ATM. *IETF Internet Draft: draft-ieft-issll-atm-support-03.txt*, July 1997.

[3] F. Bonomi, K. Fendick, and N. Yin. ABR Point-to-Multipoint Connections. *ATM Forum/95-0974R1*, August 1995.

[4] A. Charny, D. D. Clark, and R. Jain. Congestion Control With Explicit Rate Indication. *IEEE International Conference on Communications*, 1995.

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of A Fair Queueing Algorithm. *ACM SIGCOMM'89*, pages 3–12, 1989.

[6] S. Golestani. A Self-Clocked Fair Queueing Scheme for Broadband Applications. *Proceedings of IEEE INFOCOM'94*, pages 636–646, June 1994.

[7] E. Knightly, D. Wrege, J. Liebeherr, and J. Zhang. Fundamental Limits and Tradeoffs for Providing Deterministic Guarantees to VBR Video Traffic. *ACM SIGMETRICS'95*, pages 275–286, May 1995.

[8] S. Lu, K.-W. Lee, and V. Bharghavan. Adaptive Service in Mobile Computing Environments. *IWQoS'97*, May 1997.

[9] A. Parekh and R. Gallager. A Generalized Processor Sharing Approach to Flow Control - The Single Node Case. *ACM/IEEE Transactions on Networking*, pages 344–357, June 1993.

[10] H. Zhang. Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10), October 1995.

[11] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet-Switched Networks. *ACM Transactions on Computer Systems*, 9(2):101–123, May 1991.