UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEPARTAMENT D'ENGINYERIA TELEMÀTICA

TESI DOCTORAL
EN ENGINYERIA TELEMÀTICA

Fair Allocation of Network Resources for Internet Users

Director de la tesi:
Prof. Sebastià Sallent

Autor:
Albert Banchs

Novembre 2001

To Paola, Francesc, Guillermina and Carles

Contents

List of Figures

Chapter 1

Introduction

In a commercial Internet, the traffic behavior is determined by the contracts between the ISPs and the users, where a user can be a dial-up user, or one corporate network or a group of individual customers or networks. Since the user is the entity with whom the contract is signed, it should also be the unit to which network resources are allocated. However, while much research in the past has been directed to fair resource allocations for flows (see e.g. maxmin fairness [1] and proportional fairness [2][1]), much less effort has been invested on fair allocation of resources for users. The work done in this thesis tries to fill this gap: we study how to share fairly the network resources among users, when a user can possibly send several flows through different paths. Hereafter we call the concept of fairly distributing resources among users *user fairness.*

The first issue that has to be solved in order to allocate resources among users fairly is to define what network resources are. Network resources are often identified with bandwidth, but there are some applications, like e.g. voice over IP, whose performance is not determined by bandwidth alone but also by delay. We conclude that the definition of network resources is application-specific.

In this thesis we follow [4] to classify applications according to their different requirements. In [4], applications are divided in two groups: elastic applications and real-time applications. Examples of elastic applications are file transfer, electronic mail and remote terminal, and examples of real-time applications are link emulation, audio and video. Following this classification, in this thesis we deal with the problem of fairly sharing resources among users step by step (i.e. we start with a basic problem and then we incrementally extend it until solving the whole problem).

[1]In [3] we provide an overview on this research.

We first concentrate on the problem of sharing **equally** resources among users in a fair way in a **single domain** with only **elastic traffic**. In Chapter 2 we provide a solution to this problem based on the concepts of *utility* and *welfare* developed in the field of political science and political economics. We subdivide the problem in two subproblems: 1) achieve fairness with respect to the utility experienced by the different users (*inter-user fairness*) and 2) achieve fairness with respect to the utility experienced by the different flows of a user (*intra-user fairness*). The *user maxmin fairness* criterion we propose in Chapter 2 is the result of combining the two *welfare* functions that solve these two subproblems.

Along with the *user maxmin fairness* criterion, in Chapter 3 we propose a mechanism to implement it: the *User Fair Queuing* (UFQ) mechanism. In UFQ, a user is allowed to assign any label values to his packets to indicate their relative priority. At the ingress, an algorithm is used to control these labels assigned by the user. We show mathematically that: (a) the proposed label control does not allow the asymptotic throughput of a user to exceed its fair rate, and (b) if users label their packets in order to maximize their level of satisfaction or utility, then the resulting bandwidth allocation is *user maxmin fair*. The fact that neither admission control nor signaling are required in UFQ strongly contributes to its simplicity. The fact that no per-user state is kept at core nodes makes the proposed mechanism scalable. The contents of Chapters 2 and 3 have been published in [3, 5].

In Chaper 4 we extend the problem solved in Chapters 2 and 3 by adding **service differentiation** and **several domains**. We propose a network architecture for the Internet: the *User Fair Differentiation* (UFD) architecture, which extends the UFQ mechanism in such a way that its good features for resource sharing are preserved. Service Differentiation in UFD is based on the *Proportional Differentiation Model for bandwidth*, which states that the bandwidth allocated to a user should be proportional to the *share* contracted by him. We show that this model fits the requirements of service differentiation for elastic traffic. This part of the thesis has been published in [6, 7].

Chapter 5 extends the problem solved in the previous chapters by adding **real-time traffic** support. The real-time traffic extension proposed is based on the *Step Differentiation Model for delay*, which states that non-dropped real-time packets should reach their destination with a very low delay. We show that this model fits the requirements of real-time traffic. In order to account for the higher requirements of real-time traffic with respect to delay, in the proposed extension real-time traffic is assigned a higher price than elastic traffic. The part of the thesis corresponding to this chapter has been published in [8].

The above chapters define the architecture that solves the problem of fairly allocating resources among users. Chapters 6 and 7 provide extensions to this architecture for multicast traffic and wireless access respectively.

For the **multicast traffic** part, we take up previous work on bandwidth allocation for unicast and multicast flows: the Logarithmic Receiver Dependent (*logRD*) policy. This policy encourages the use of multicast traffic delivery while providing a good level of fairness between unicast and multicast. In Chapter 6 we propose an extension of the UFD architecture that implements the *logRD* policy. In addition, the proposed extension supports layered multicast transmissions. This part of the thesis has been published in [9, 10, 11].

In Chapter 7 we propose an extension of the UFD architecture for **Wireless LAN**. The proposed extension allocates resources in the Wireless LAN following the same principles as the architecture described in the previous chapters for wired networks. The challenge in Wireless LAN is that we do not have all packets in a centralized queue, like in wired links, but we have them distributed in the wireless hosts. Therefore, we need a MAC mechanism capable of providing the desired scheduling. The proposed wireless architecture has been published in [12, 13, 14, 15].

The architecture proposed in this thesis has been implemented in a Linux PC platform. In Chapter 8 we report on implementation experiences. This part has been published in [9, 16].

Publications of the author highly related with the content of this thesis can be found in [17, 18, 19, 20, 21, 22, 23, 24].

Chapter 2

User Fairness

User fairness in the current best-effort Internet is based on *TCP fairness* [25], which distributes network resources among users on a flow basis: with TCP, each flows receives a throughput inversely proportional to its round-trip delay. However, not all flows in the Internet use the TCP protocol and it is relatively easy for non-adaptive sources to gain greater shares of network bandwidth and thereby starve other, well-behaved, TCP friendly sources. For that reason the TCP fairness solution relies on TCP friendly behavior of the applications in order to fairly share the network resources. In addition, the fact that TCP fairness assigns resources on a flow basis makes the amount of resources received by each user dependent on the number of flows sent, which may lead to an unfair overall distribution of the resources.

One approach for user fairness that overcomes the above mentioned problems of TCP fairness is the *pre-allocation of resources on a link-basis*. This is the fairness concept behind fair user-based queuing schemes, such as Weighted Fair Queuing (WFQ) [26] and Class-based Queuing (CBQ) [27]. These queuing algorithms distribute the link bandwidth among users in a fair way, independent of the number of flows that each user is sending through this link and the aggressiveness of the sources. One of the remaining drawbacks of these approaches is that since they work on a local basis, they cannot ensure a fair distribution of the overall network resources.

In the last few years, architectures for providing Quality of Service (QoS) in the Internet have been the focus of extensive research. These research efforts have identified two fundamentally different approaches for QoS: Integrated Services (IntServ) [28] and the Differentiated Services (DiffServ) [29]. The fairness concept behind these architectures is the *resource allocation on demand*. With IntServ and DiffServ, users request a certain amount of network resources; the request is processed by an admission control entity in order to determine whether the available resources can satisfy the request, and the user is notified of the acceptance or rejection of his request. One of the important challenges of these architectures is precisely how to perform this admission control. With IntServ this was done in a flow basis, which leads to unscalability, and with DiffServ admission control is still an open issue (in [20] and [21] we study this issue for one-to-one and one-to-any services, respectively).

In this chapter we introduce a new concept for user fairness that overcomes the drawbacks of the above mentioned approaches. The proposed concept distributes resources equally among users taking into account the overall usage of the network resources, while admission control is avoided. The goal is to provide two users that pay a same price with the same amount of resources, independent of the number of flows and links used by each user and the aggressiveness of the sources. As discussed in the Introduction, we focus on elastic traffic and a single domain.

2.1 Review on Fairness Concepts in Computer Networks

In the following we provide a review on fairness concepts in computer networks. These concepts will be applied in Section 2.2 to the problem of fairly allocating resources for users, resulting in the *user maxmin fairness* criterion.

The concept of *fairness* has been studied in various scientific areas. Most thorough and theory-based approaches arose from the field of political science and political economics. In this field, the concepts of *utility* [30] and *welfare* [31] functions were developed for the purpose of defining *fairness*. In this section we review this theory from the computer network's viewpoint. In [3] we provide a more extensive description of the concepts explained in this section.

2.1.1 Utility function

In order to express the user's satisfaction with the service delivered to him by the network, network performance must not be measured in terms of network-centric quantities like throughput, packet drops or delay, but should be rather evaluated in terms of the degree to which the network satisfies the service requirements of each user's applications. For instance, if a particular application cares more about throughput than delay, or vice-versa, the network service to that application should be evaluated accordingly.

Utility functions in computer networks [4] formalize the above notion of network performance. Let s_i describe the service delivered to the i'th

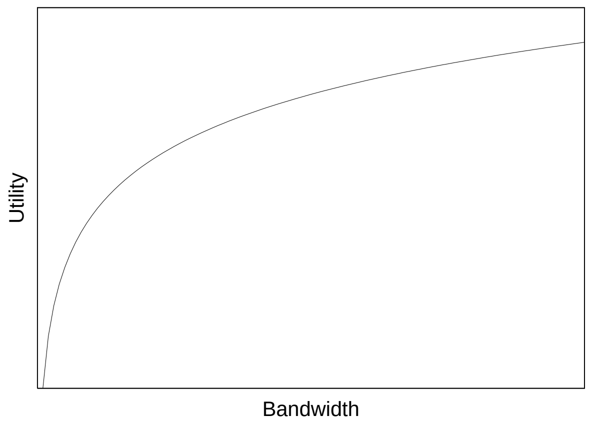

Figure 2.1: Utility function of an elastic traffic flow.

application or user; s_i contains all the relevant measures (delay, throughput, packet drops, etc.) of the delivered service. Then, the utility function u_i maps the service delivered s_i into the performance of the application; increasing u_i reflects increasing application performance. The utility function, thus, describes how the performance of an application depends on the delivered service.

In the following, we elaborate on the utility function for elastic traffic, i.e. the traffic type on which we are concerned in this chapter. Examples of elastic applications are file transfer, electronic mail and remote terminal. These applications are tolerant of delays and their satisfaction is basically measured in terms of bandwidth. Therefore, bandwidth is the only relevant measure for this traffic type and the only one that will be considered in Chapters 2 and 3.

Elastic applications experience a marginal rate of performance enhancement as bandwidth is increased, so their utility function is strictly concave everywhere. Following [2], in this thesis we use the logarithmic function to represent the utility of elastic traffic (see Figure 2.1). Thus, the utility of an elastic flow i will be

$$u_i(r_i) = log(r_i) \tag{2.1}$$

where r_i is the flow's throughput.

2.1.2 Welfare function

The basic problem of *welfare* is to determine which of the feasible resource allocations should be selected. For this purpose, a *Welfare function* $W(u_1, u_2,$

$\ldots, u_n)$ that aggregates the individual utility functions u_i is defined. The resource allocation selected (called the *fair resource allocation*) is then the one that maximizes the welfare function:

$$max(W(u_1, u_2, \ldots, u_n)) \tag{2.2}$$

For different purposes, different welfare functions exist, each corresponding to a different fairness criterion. A fairness criterion, thus, is defined by the welfare function that it maximizes. The most widely used fairness criteria in computer networks are *maxmin fairness* and *proportional fairness.*

Maxmin fairness

Maxmin fairness [1] is the most popular fairness concept in computer networks. This fairness criterion corresponds to the welfare function:

$$W(u_1, u_2, \ldots, u_n) = min(u_1, u_2, \ldots, u_n) \tag{2.3}$$

Maxmin fairness, thus, yields a solution $u = (u_1, u_2, \ldots, u_n)$ for $max(min$ $(u_1, u_2, \ldots, u_n))$. A maxmin fair allocation has the property that for all i, u_i cannot be increased without simultaneously decreasing u_j for some j with $u_j \leq u_i$.

The idea behind maxmin fairness is to distribute resources as equally as possible among the competing entities. As a consequence, with this criterion the most poorly treated entities are given the greatest possible allocation.

Proportional fairness

The proportional fairness criterion [2] is becoming increasingly popular in the field of computer networks. A proportional fair allocation is the solution to the welfare maximization problem with the welfare function sum of utilities:

$$W(u_1, u_2, \ldots, u_n) = \sum_i u_i \tag{2.4}$$

and with the individual utility functions u_i of Equation 2.1.

Proportional fairness, thus, yields a solution u for $max(\sum_i u_i)$. A proportional fair allocation has the property that for any other feasible allocation u^*, the aggregate of proportional changes is zero or non-negative, i.e.

$$\sum_i (u_i^* - u_i)/u_i \leq 0 \tag{2.5}$$

The idea behind proportional fairness is to maximize the overall performance. With proportional fairness, a worse treated entity may see its utility decreased if this allows a large enough increase to an already better treated entity.

Weighted Fairness

Both maxmin and proportional fairness criteria can be generalized on introducing weights W_i associated with each entity as a means to express the relative value of this entity for the system [32]. With weighted fairness, the utility received by an entity in the fair allocation will increase with its associated weight W_i.

The introduction of weighting leads to the following welfare function for *weighted maxmin fairness*

$$W(u_1, u_2, \ldots, u_n) = min(u_i(r_i/W_i)) \tag{2.6}$$

and *weighted proportional fairness*

$$W(u_1, u_2, \ldots, u_n) = \sum_i W_i \cdot u_i \tag{2.7}$$

Weighted maxmin fairness aims at distributing resources among the competing entities proportionally to their weights. Weighted proportional fairness aims at maximizing the overall performance when some entities have a higher value than others.

2.2 User Maxmin Fairness

User Fairness deals with the allocation of bandwidth among users, when a user may send one or more flows, possibly through different paths. Each flow i of user u experiences a certain utility u_i, which depends on its allocated bandwidth r_i as defined in Equation 2.1.

Following the fairness concepts explained in the previous section, a user fair allocation is the one that maximizes the welfare function W that aggregates the individual utilities u_i, $W(u_i)$. In this section we study which is the appropriate welfare function for the problem of user fairness.

2.2.1 Welfare function composition

According to the definition of welfare in Section 2.1, the utility experienced by a user u, u_u, is the result of aggregating with a welfare function the utility of the flows of this user (intra-user aggregation). We call W_{intra} the welfare function that performs this intra-user aggregation:

$$u_u = \underset{i \in U}{W_{intra}}(u_i) \tag{2.8}$$

where U is the set of flows of user u.

Similarly, the total welfare experienced in the network is the result of aggregating the individual utilities of all the users in the network (inter-user aggregation). We call W_{inter} the welfare function that performs this inter-user aggregation:

$$W = \underset{\forall u}{W_{inter}}(u_u) = \underset{\forall u}{W_{inter}}(\underset{i \in U}{W_{intra}}(u_i)) \tag{2.9}$$

The bandwidth allocation for user fairness, thus, will be the one that maximizes the welfare function $W_{inter}(W_{intra}(\cdot))$, choosing the appropriate functions for W_{inter} and W_{intra}. We will choose the functions W_{inter} and W_{intra} according to the goal of providing a good level of inter and intra user fairness as described in the following.

2.2.2 Inter and Intra User fairness

We say that a bandwidth allocation is *inter-user fair* if the network bandwidth is fairly distributed among the different users. Similarly, we say that a bandwidth allocation is *intra-user fair* if the bandwidth assigned to a user is fairly distributed among his flows. Inter and intra user fairness are better illustrated in the example of Figure 2.2. In this example we have a network with three links (a,b,c) and three users (1,2,3). The first user (user 1) is sending three flows, one through each of the links (a,b,c), the second (user 2) is sending two flows, one through link a and the other through link b, and the third user (user 3) is sending only one flow through link c. All links have a capacity normalized to 1.

An inter-user fair allocation for the above scenario would be the following:

$$\begin{array}{rr} r_{1a} = 1/2 & r_{2a} = 1/2 \\ r_{1b} = 1/2 & r_{2b} = 1/2 \\ r_{1c} = 0 & r_3 = 1 \end{array}$$

Note that in the above allocation all users get the same total bandwidth (1 unit of capacity) and therefore the allocation is inter-user fair. However, if we look on how the bandwidth allocated to user 1 is distributed among his flows, we observe an extreme degree of intra-user unfairness. User 1 will most probably not be satisfied with the above allocation, since one of his flows is totally starved.

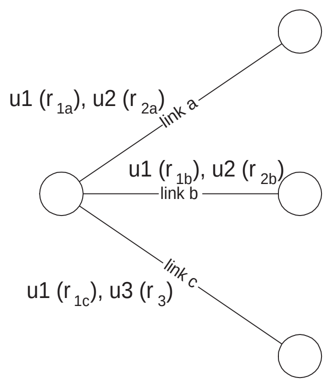

Figure 2.2: Example of inter and intra user fairness.

Another possible allocation that corrects the intra-user unfairness of the first one is the following:

$$\begin{aligned} r_{1a} &= 1/2 \quad r_{2a} = 1/2 \\ r_{1b} &= 1/2 \quad r_{2b} = 1/2 \\ r_{1c} &= 1/2 \quad r_{3} = 1/2 \end{aligned}$$

The above distribution provides a perfect level of intra-user fairness, since for each user, all his flows experience the same throughput. However, the level of inter-user fairness is poor: in link c, users 1 and 3 are allocated the same bandwidth, even though user 1 is using more network resources than user 3 in total. User 3 will most probably not be satisfied with this allocation.

We conclude that a *user fair* allocation should provide a good level of both inter and intra user fairness. In the following we study which welfare functions W_{inter} and W_{intra} to choose in order to achieve this goal.

2.2.3 Definition

The goal of inter-user fairness is to treat the different users as equally as possible. In Section 2.1.2 we have argued that the fairness criterion that best meets this goal is the *maxmin fairness* criterion. As a consequence, we have chosen to use the welfare function minimum for the aggregation of the utilities of the different users:

$$W_{inter}^{fairness} = min(u_u) \;\; \forall \text{ user } u \text{ in the network} \tag{2.10}$$

The goal of intra-user fairness is to allocate the bandwidth received by a user among his flows as equally as possible to the user's desired distribution.

In Section 2.1.2 we have argued that the fairness criterion that best meets this goal is the *weighted maxmin fairness* criterion. This is the criterion we have chosen for intra-user aggregation, as expressed by the following welfare function:

$$W_{intra}^{fairness} = \underset{i \in U}{min}(u_i(r_i/W_i)) \tag{2.11}$$

with the constraint

$$\sum_{i \in U} W_i = 1 \tag{2.12}$$

where U is the set of flows of user u and W_i are the normalized weights that express the relative value of flow i for its user.

The normalization of the sum of the weights of a user to 1 (Equation 2.12) comes from the necessity of being able to compare the $W_{intra}^{fairness}$ of different users. Note that with Equation 2.12, two users that get the same total bandwidth and have this bandwidth distributed proportionally to the weights W_i experience the same $W_{intra}^{fairness}$.

The combination of $W_{inter}^{fairness}$ and $W_{intra}^{fairness}$ leads to the following definition.

Definition 1 (User Maxmin Fairness) [1] *A bandwidth allocation $r = (r_1, r_2, \ldots, r_n)$ is* user maxmin fair *when it maximizes*

$$\underset{\forall i}{min}(r_i/W_i) \tag{2.13}$$

where r_i is the throughput experienced by flow i and W_i is its normalized weight.

The proposed criterion for user fairness leads to the following allocation for the example of Figure 2.2:

$$\begin{array}{ll} r_{1a} = 2/5 & r_{2a} = 3/5 \\ r_{1b} = 2/5 & r_{2b} = 3/5 \\ r_{1c} = 1/4 & r_3 = 3/4 \end{array}$$

which is a good tradeoff between the inter and intra user fair allocations given in Section 2.2.2.

[1]Note that in the special case when all users are sending just one flow, the user maxmin fairness criterion coincides with the well accepted maxmin fairness criterion for flows.

2.3 User utility

The level of satisfaction of a user depends on the overall performance of his flows, where some of his flows may have a higher relative value than others. In Section 2.1.2 we have argued that the welfare function that best expresses this level of satisfaction of a user is the weighted sum function, corresponding to the *weigthed proportional fairness* criterion. In the following definition of user utility we have used this welfare function to perform intra-user aggregation. The definition of user utility will be used later in the thesis to model the user behavior.

Definition 2 (User Utility) *The utility of user u, whose flows experience a throughput equal to r_i, is given by*

$$u_u = W_{intra}^{utility}(\cdot) = \sum_{i \in U} W_i \cdot u_i(r_i) = \sum_{i \in U} W_i \cdot log(r_i) \qquad (2.14)$$

2.4 Summary

User Fairness aims at a fair distribution of network resources among users. The need for user fairness is motivated by the fact that the user is commonly the entity to which pricing schemes apply; as a consequence, the user should also be the unit to which network resources are assigned.

However, while much effort has been invested in the past for the definition of fairness among flows, much less effort has been spent to address fairness among users. A user is an entity that may possibly send different flows through different paths.

In this chapter we have proposed the *user maxmin fairness* criterion with the goal of fairly distributing the network bandwidth among users when these users are sending elastic traffic.

Chapter 3

User Fair Queuing

In this chapter, we propose a network architecture, *User Fair Queuing* (UFQ), that provides *user maxmin fairness* as defined in the previous chapter.

The proposed scheme avoids keeping per-flow or per-user state in the core and is inspired on previous work done in the context of core-stateless fair allocation of bandwidth among flows [33, 34, 35, 36]. While these proposals differ in details, they are all similar at the architectural level. Per-flow state at the core is avoided by having each packet header carry some additional state information, the *label*, which is initialized by the ingress node of the network. Then, core nodes use this information carried by the packet to decide whether in case of congestion an arriving packet should be enqueued or dropped.

UFQ is implemented in three steps: *user labeling*, *ingress label control* and *core dropping* (see Figure 3.1). In the first step (*user labeling*), the user assigns labels to his packets based on the sending rates of his flows and their weights (i.e. per-flow state is required). The second step (*ingress label control*) is performed at the ingress of the network. In this step, the labels assigned by the user are processed, and in case the user is labeling his packets with more resources than he should, packets are relabeled. The algorithm proposed for the ingress label control only requires keeping per-user state (i.e. it avoids per-flow state). Finally, the third step is performed at core nodes, where in case of congestion packets are dropped depending on their label. Since the *core dropping* is performed without keeping per-user or per-flow state, the proposed architecture scales with the number of users.

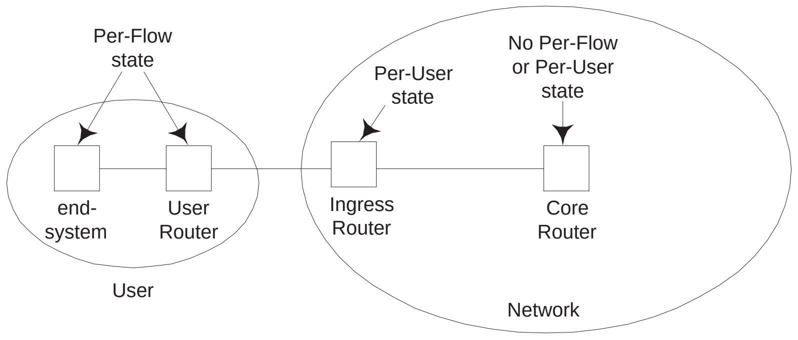

Figure 3.1: UFQ architecture.

3.1 User labeling

At the user network, packet k of flow i is labeled with:

$$L_k = \frac{r_i^{send}}{W_i} \tag{3.1}$$

where W_i is the weight of flow i as defined in Section 2.2 and r_i^{send} is the flow's sending rate.

For the estimation of the sending rate of flow i we use the same exponential averaging formula as in [33]. Using an exponential averaging gives more accurate estimation for bursty traffic, even when the packet inter-arrival time has significant variance.

Specifically, let t_k and l_k be the arrival time and length of the k^{th} packet of flow i. The estimated sending rate of flow i, r_i^{send}, is updated for every new packet k sent by flow i:

$$(r_i^{send})_k = (1 - e^{-(T_k/K)})\frac{l_k}{T_k} + e^{-(T_k/K)} \cdot (r_i^{send})_{k-1} \tag{3.2}$$

where $T_k = t_k - t_{k-1}$ and K is a constant. Following the rationale discussed in [33], we set $K = 100ms$.

3.2 Ingress Label Control

The probability of dropping a packet of flow i should decrease with the relative value of this flow for its user (W_i) and should increase with the flow's sending rate (r_i^{send}). As a consequence, the lower the value of the packet's label L_k, the better treatment this packet should be given. If there was no control on the labels assigned by a user, a user could exploit the system by assigning to his packets labels with lower values than Equation 3.1. The goal of the *ingress label control* is not to allow a user to benefit from labeling his packets with too low values.

Note that keeping per-flow information at the ingress (namely, W_i and r_i^{send}) label control could easily enforce Equation 3.1. However, this would introduce a considerable complexity at ingress nodes and would require the user to signal the values of W_i to the ingress. The algorithm we propose for label control avoids per-flow state and only requires per-user state.

The ingress label control algorithm is based on the observation that the following equality holds for a user who is labeling his packets as in (3.1):

$$S_u = \frac{\underset{k\in U_k}{avg}\left(\frac{l_k}{L_k}\right)}{\underset{k\in U_k}{avg}\left(l_k\right)} r_u^{send} = \sum_{i\in U_i} W_i = 1 \tag{3.3}$$

where U_k is the set of packets of user u, U_i is the set of his flows, l_k is the length of packet k, L_k is its label and S_u (the state of user u at the ingress) is defined by the first equality of the above equation.

Having too low labels would lead to S_u being larger than 1. In order to avoid too low labels, we enforce

$$S_u \leq 1 \tag{3.4}$$

S_u is estimated using the following formula upon receiving the k^{th} packet of user u:

$$(S_u)_k = (1 - e^{-(\frac{l_k}{r_u^{send}.K})})\frac{r_u^{send}}{L_k} + e^{-(\frac{l_k}{r_u^{send}.K})} \cdot (S_u)_{k-1} \tag{3.5}$$

where r_u^{send} is estimated using Equation 3.2. The reason for using this estimation for S_u is that it allows us to bound the excess service that a user can achieve, as discussed in Section 3.4.

The *ingress label control* enforces Equation 3.4 in the following way: if the arriving packet of a user has a label L_k that would lead to $(S_u)_k > 1$, then we relabel the packet with a new label $L_k^{new} > L_k$ such that $(S_u)_k = 1$. Thus,

$$L_k^{new} = max\left(L_k, \frac{(1 - e^{-(\frac{l_k}{r_u^{send}.K})})r_u^{send}}{1 - e^{-(\frac{l_k}{r_u^{send}.K})} \cdot (S_u)_{k-1}}\right) \tag{3.6}$$

Note that a user who is labeling his packets as in (3.1) will not have his packets relabeled, since according to Equation 3.3, the labels L_k of this user will never lead to $(S_u)_k$ greater than 1.

Note that the above algorithm for the ingress label control only requires to keep per-user state at the ingress (namely, two values have to be stored for each user: S_u and r_u^{send}). The effectiveness of the proposed scheme will be discussed in Sections 3.4 and 3.5.

3.3 Core dropping

In a congested link in which there is not enough bandwidth to serve all incoming packets, some packets must be dropped. Our goal is to drop packets in such a way that the resulting bandwidth distribution is *user maxmin fair*.

In UFQ, packets in a congested link l are dropped depending on their label with the following probability:

$$d_k = \begin{cases} 0 & L_k \leq L_{fair} \\ 1 - \frac{L_{fair}}{L_k} & L_k > L_{fair}, L_k^{new} = L_{fair} \end{cases} \tag{3.7}$$

where d_k is the probability of dropping packet k, L_k is its label and L_{fair} is the *fair label* estimation in the congested link. Note that the non-dropped packets of a flow that experiences losses in a link are relabeled with a new label L_k^{new}.

The following theorem binds the algorithms proposed for user labeling and core dropping with the *user maxmin fairness* criterion of Section 2.2.

Theorem 1 *The bandwidth allocation resulting from the user labeling of (3.1) and the core behavior of (3.7) is* user maxmin fair.

Proof: The core dropping of (3.7) leads to the following value for the outgoing rate at link l of a flow i competing for bandwidth in this link:

$$r_i^{out} = r_i^{in}(1 - d_k) = r_i^{in}\frac{L_{fair}}{L_k} \tag{3.8}$$

Substituting the relabeling $L_k^{new} = L_{fair}$ in the above equation leads to the following relationship between the outgoing rate of flow i at link l, r_i^{out}, and the label values of its outgoing packets, L_k^{new}:

$$\frac{r_i^{out}}{L_k^{new}} = \frac{r_i^{in}}{L_k} \tag{3.9}$$

According to the above equation the relationship between the rate of a flow and its label values is kept constant when crossing a link. Thus, the initial value of this relationship (*user labeling*, Equation 3.1) will be preserved in all links:

$$\frac{r_i^{out}}{L_k^{new}} = W_i \tag{3.10}$$

Combining Equations 3.8 and 3.10 leads to the following outgoing rate of flow i at link l:

$$r_i^{out} = W_i \cdot L_{fair} \tag{3.11}$$

From the above equation it can be observed that with the dropping algorithm of Equation 3.7, bandwidth in a congested link is distributed among two competing flows i and j proportionally to their weights:

$$\frac{r_i^{out}}{W_i} = \frac{r_j^{out}}{W_j} = L_{fair} \tag{3.12}$$

If we increase r_i for some flow i crossing link l, this will force to decrease r_j for some other flow j that also crosses link l. Hence, for all i r_i cannot be increased without decreasing $\frac{r_j}{W_j}$ for some j for which $\frac{r_j}{W_j} \leq \frac{r_i}{W_i}$. As a consequence, the minimum $\frac{r_i}{W_i}$ is maximized, which leads to *user maxmin fairness.* ■

One remaining challenge is the estimation of the *fair label* L_{fair}. L_{fair} should be such that, in case of congestion, the rate of enqueued packets, F, equaled the link's capacity C. For scalability reasons, L_{fair} should be estimated without storing any per-user or per-flow information at the core nodes. Different solutions to the problem of estimating L_{fair} without core state have been proposed in [33, 34, 36, 37]. The algorithm that we have used is the one proposed in [33].

To compute L_{fair}, [33] keeps two aggregate variables: A, the estimated aggregated arrival rate, and F, the estimated rate of the accepted traffic. Then, L_{fair} is updated every K time units according to the following algorithm:

```
if $A \geq C$ then {link congested}
    $(L_{fair})_{new} = (L_{fair})_{old} \cdot C/F$
else {link uncongested}
    $(L_{fair})_{new} =$ largest $L_i$ observed
end if
```

The UFQ architecture resulting from the *user labeling*, *ingress label control* and *core dropping* algorithms is described in pseudocode in Algorithm 1 and illustrated in Figure 3.2.

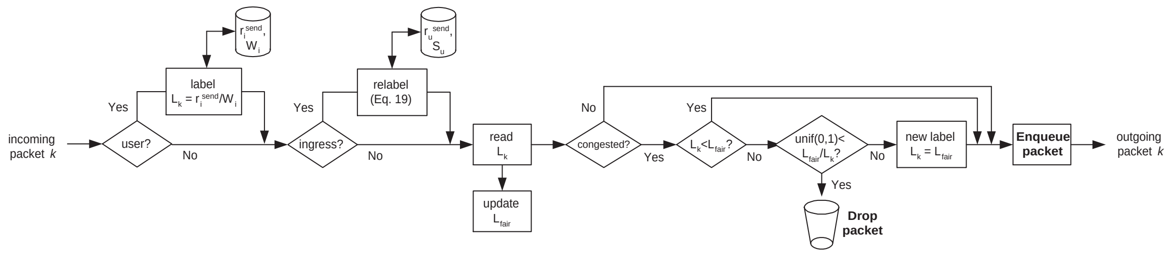

Figure 3.2: UFQ algorithm.

Algorithm 1 UFQ pseudocode

```
User labeling:
on receiving packet $k$
  $r_i^{send} = (1 - e^{-(T_k/K)})\frac{l_k}{T_k} + e^{-(T_k/K)} r_i^{send}$
  $L_k = \frac{r_i^{send}}{W_i}$
  write_label($L_k$)

Ingress Label Control:
on receiving packet $k$
  read_label($L_k$)
  $r_u^{send} = (1 - e^{-(T_k/K)})\frac{l_k}{T_k} + e^{-(T_k/K)} r_u^{send}$
  $L_k = max\left(L_k, \frac{(1-e^{-(\frac{l_k}{r_u^{send} \cdot K})}) r_u^{send}}{1-e^{-(\frac{l_k}{r_u^{send} \cdot K})} \cdot S_u}\right)$
  $S_u = (1 - e^{-(\frac{l_k}{r_u^{send} \cdot K})})\frac{r_u^{send}}{L_k} + e^{-(\frac{l_k}{r_u^{send} \cdot K})} \cdot S_u$
  write_label($L_k$)

Core dropping:
on receiving packet $k$
  read_label($L_k$)
  estimate $L_{fair}$
  $prob = max(0, 1 - \frac{L_{fair}}{L_k})$
  if $prob >$ unif_rand(0, 1) then
    drop(packet $k$)
  else
    enqueue(packet $k$)
  end if
  if $prob > 0$ then
    write_label($L_{fair}$)
  end if
```

3.4 Ingress Label Control and Excess Service

The service data received by a user who is labeling his packets as in (3.1) during a time interval T is:

$$F = T \sum_{i \in U} r_i = T \sum_{i \in U} W_i \cdot L^i_{fair} \tag{3.13}$$

where L^i_{fair} is the fair label of flow i's bottleneck and

$$\sum_{i \in U} W_i = 1 \tag{3.14}$$

F as defined above is the service to which a user is entitled. We call any amount above this the *excess service.*

The ingress label control presented in Section 3.2 has been designed with the goal of avoiding that a user can obtain more service than he is entitled to. In this section we study how well the scheme we have proposed meets this goal.

We cannot study the above issue with full generality, but we can analyze a simplified situation where the *fair label* L_{fair} of all links is held fixed. In addition, we assume that when a packet arrives a fraction of that packet equal to the flow's forwarding probability is transmitted.

Theorem 2 (at the end of this section) gives an upper bound to the excess service received by a user in this idealized setting. This bound is independent of the arrival process, the incoming labels and the time interval. The bound does depend crucially on the maximal rate R at which user's packets can arrive at the ingress (limited, for example, by the speed of the user's access link); the smaller this rate R the tighter the bound.

By bounding the excess service, we show that in the idealized setting the asymptotic throughput received by a user cannot exceed the throughput he is entitled to. Thus, users can only exploit the system over short time scales; the ingress label control limits their throughput effectively over long time scales.

Lemma 1 *Consider a user sending all his flows through one bottleneck link with a constant* fair label L_{fair}. *Then, the excess service F_{excess} received by this user, that sends at a rate no larger than R, is bounded above by*

$$F_{excess} < l_{max} + L_{fair} \cdot K \left(2 + ln \frac{R}{L_{fair}} \right) \tag{3.15}$$

where l_{max} represents the maximum length of a packet and K is the averaging constant of Equation 3.2.

Proof: Without loss of generality assume that during the target time interval T the user sends exactly n packets. Let t_k be the arrival time of the k^{th} packet, l_k its length and L_k its label. Since in case the label L_k is smaller than L_{fair} the packet is always forwarded, and in case it is larger it is forwarded with probability L_{fair}/L_k, the service received by the user during the interval can be expressed as

$$F = \sum_{k=1}^{n} min\left(l_k, l_k \frac{L_{fair}}{L_k}\right) \tag{3.16}$$

The problem of bounding the service received by a user can be reformulated as to maximizing F.

Assume that the user becomes active for the first time at t_1. Let t_m be the first time when his rate estimator r^{send} exceeds for the first time L_{fair} and m the number of packets sent during the interval (t_1, t_m). We first bound the service received in the interval (t_1, t_m), which we denote by F_1.

With the restriction imposed by the ingress label control

$$S_k = (1 - e^{-(\frac{l_k}{r_k^{send}.K})})\frac{r_k^{send}}{L_k} + e^{-(\frac{l_k}{r_k^{send}.K})} \cdot S_{k-1} \leq 1 \tag{3.17}$$

it can be easily shown that F is maximized with $L_k \geq L_{fair}$ for $1 \leq k \leq n$, since a packet with $L_k < L_{fair}$ will contribute to F as much as with $L_k = L_{fair}$ but with the restriction of (3.17) will force larger labels on the other packets, resulting in a lower F. Then,

$$max(F) = \max_{L_k \geq L_{fair}} \left(\sum_{k=1}^{n} l_k \frac{L_{fair}}{L_k}\right) \tag{3.18}$$

Let us define

$$G = \sum_{k=1}^{n} l_k \frac{L_{fair}}{L_k} \tag{3.19}$$

Isolating l_k/L_k in Equation 3.17

$$\frac{l_k}{L_k} = \frac{S_k - e^{-(\frac{l_k}{r_k^{send}.K})} S_{k-1}}{1 - e^{-(\frac{l_k}{r_k^{send}.K})}} \frac{l_k}{r_k^{send}} \ , \ 1 \leq k \leq n \tag{3.20}$$

and substituting it in G leads to

$$\frac{\partial G}{\partial S_k} = \frac{\frac{l_k}{r_k^{send}}}{1 - e^{-(\frac{l_k}{r_k^{send}.K})}} - \frac{\frac{l_{k+1}}{r_{k+1}^{send}} e^{-(\frac{l_{k+1}}{r_{k+1}^{send}.K})}}{1 - e^{-(\frac{l_{k+1}}{r_{k+1}^{send}.K})}} \ , \ 1 \leq k < n \tag{3.21}$$

and

$$\frac{\partial G}{\partial S_n} = \frac{\frac{l_n}{r_n^{send}}}{1 - e^{-(\frac{l_n}{r_n^{send} \cdot K})}} \tag{3.22}$$

Since $x/(1 - e^{-x}) \geq 1$ and $xe^{-x}/(1 - e^{-x}) \leq 1$ for any $x \geq 0$, we have

$$\frac{\partial G}{\partial S_k} \geq 0 \ , \ 1 \leq k \leq n \tag{3.23}$$

We now focus on the interval (t_1, t_m). From Equation 3.23 we conclude that F is maximized when S_k achieves its maximum value for $1 \leq k \leq m$. The maximum S_k is given when the values of the labels L_k are minimum, subject to the ingress label control of Equation 3.17 and the condition $L_k \geq L_{fair}$ given before.

Since the minimum value of L_k allowed by the last condition, $L_k = L_{fair}$, satisfies Equation 3.17, we have that F is maximized with $L_k = L_{fair}$ for $1 \leq k \leq m$. In this case,

$$F_1 = \sum_{k=1}^{m} l_k \tag{3.24}$$

Thus, the problem of bounding F_1 can be reformulated as to find the maximum $\sum_{k=1}^{m} l_k$ under the restriction that imposes the assumption that in the interval (t_1, t_m) the rate estimator r_k^{send} does not exceed the *fair label* L_{fair}. The solution to this problem is given by Lemma 1 of [38], which states that the maximum service received in the above conditions is achieved when $r_k^{send} = L_{fair}$ for $1 \leq k \leq m$ and is bounded above by:

$$F_1 < l_{max} + L_{fair} \cdot K + (t_m - t_1)L_{fair} \tag{3.25}$$

As said above, t_m is the time when the rate estimator exceeds for the first time L_{fair}. If such time t_m does not exist, according to the above the *excess service* is bounded by $l_{max} + L_{fair} \cdot K$ which concludes the proof for this case. In the following we consider the case when t_m exists.

Next, we show that the service received by a user is maximized when $r^{send}(t) \geq L_{fair} \ \forall \ t > t_m$. The proof goes by contradiction. Assume there is an interval (t', t'') such that $t' > t_m$ and $r^{send}(t) < L_{fair}$. Then, using an identical argument as in the interval (t_1, t_m), it can be easily shown that the service received by the user is maximized when $r^{send}(t) = L_{fair} \ \forall \ t \in (t', t'')$.

Now, we bound the service received in the remaining interval (t_m, t_n), F_2, for which we have shown that $r^{send}(t) \geq L_{fair}$.

$$F_2 = \sum_{k=m+1}^{n} min\left(l_k, l_k \frac{L_{fair}}{L_k}\right) \tag{3.26}$$

Taking only the second term of the minimum gives an upper bound to F_2,

$$F_2 \leq G_2 = \sum_{k=m+1}^{n} l_k \frac{L_{fair}}{L_k} \tag{3.27}$$

Using the same argument as for G, it can be easily shown that

$$\frac{\partial G_2}{\partial S_k} \geq \ 0, \ m+1 \leq k \leq n \tag{3.28}$$

As a result, G_2 is maximized when S_k achieves its maximum value. The ingress label control limits S_k to 1. Hence, G_2 will be maximized when

$$S_k = 1 \ , \ m+1 \leq k \leq n \tag{3.29}$$

Combining the above with Equation 3.17 leads to

$$\frac{l_k}{L_k} = \frac{l_k}{r_k^{send}} \ , \ m+2 \leq k \leq n \tag{3.30}$$

For $k = m+1$ we have (Equation 3.20 with $S_{m+1} = 1$)

$$\frac{l_{m+1}}{L_{m+1}} = \frac{1 - e^{-(\frac{l_{m+1}}{r_{m+1}^{send} \cdot K})} S_m}{1 - e^{-(\frac{l_{m+1}}{r_{m+1}^{send} \cdot K})}} \frac{l_{m+1}}{r_{m+1}^{send}} \leq \frac{1}{1 - e^{-(\frac{l_{m+1}}{r_{m+1}^{send} \cdot K})}} \frac{l_{m+1}}{r_{m+1}^{send}} \tag{3.31}$$

Further, by assuming $K \gg l_{m+1}/r_{m+1}^{send}$ we obtain

$$\frac{l_{m+1}}{L_{m+1}} \approx K \tag{3.32}$$

The combination of the above results gives the following upper bound for F_2:

$$F_2 \leq L_{fair} \cdot K + \sum_{k=m+2}^{n} l_k \frac{L_{fair}}{r_k^{send}} \tag{3.33}$$

Lemma 2 of [38] shows that the term $\sum_{k=m+2}^{n} l_k \frac{L_{fair}}{r_k^{send}}$ is bounded above by $L_{fair} \cdot K \cdot ln(R/L_{fair}) + (t_n - t_{m+2})L_{fair}$ when $r_k^{send} \geq L_{fair}$ for $m+1 \leq k \leq n$. Using this result yields

$$F_2 < L_{fair} \cdot K + L_{fair} \cdot K \cdot ln\left(\frac{R}{L_{fair}}\right) + (t_n - t_m)L_{fair} \tag{3.34}$$

Finally, combining the bounds found for F_1 and F_2 yields

$$F = F_1 + F_2 < l_{max} + L_{fair} \cdot K \left(2 + ln\frac{R}{L_{fair}}\right) + (t_n - t_1)L_{fair} \tag{3.35}$$

Since $(t_n - t_1)L_{fair}$ represents exactly the number of bits that the user is entitled to send during the interval $(t_n - t_1)$, the proof follows. ■

Theorem 2 *Consider a user sending n flows through n bottleneck links, all links with a constant* fair label L^i_{fair}. *Then, the excess service F_{excess} received by this user, that sends at a rate no larger than R, is bounded above by*

$$F_{excess} < \left(\sum_{i\in U} L^i_{fair} \cdot W_i\right) \cdot \left(\frac{l_{max}}{L^{min}_{fair}} + K\left(2 + ln\frac{R}{L^{min}_{fair}}\right)\right) \tag{3.36}$$

where U is the set of flows of the user, $L^{min}_{fair} = \min\limits_{i\in U}(L^i_{fair})$ *and*

$$\sum_{i\in U} W_i = 1 \tag{3.37}$$

Proof: From Lemma 1 it follows that

$$\begin{aligned} &max\left(\sum_{k\in T} min\left(l_k, l_k\frac{L_{fair}}{L_k}\right)\right) = \\ &= \max_{L_k\geq L_{fair}}\left(\sum_{k\in T} l_k\frac{L_{fair}}{L_k}\right) = \\ &= L_{fair} \cdot \max_{L_k\geq L_{fair}}\left(\sum_{k\in T}\frac{l_k}{L_k}\right) < \\ &< l_{max} + L_{fair} \cdot K\left(2 + ln\frac{R}{L_{fair}}\right) + (t_n - t_1)L_{fair} \end{aligned} \tag{3.38}$$

which yields

$$\max_{L_k\geq L_{fair}}\left(\sum_{k\in T}\frac{l_k}{L_k}\right) < \frac{l_{max}}{L_{fair}} + K\left(2 + ln\frac{R}{L_{fair}}\right) + (t_n - t_1) \tag{3.39}$$

The service received by a user is

$$F = \sum_{k\in T} min\left(l_k, l_k\frac{L^i_{fair}}{L_k}\right) \tag{3.40}$$

where L^i_{fair} is the *fair label* of packet k's bottleneck link and T is the set of packets sent by the user in the interval (t_1, t_n).

Using the same argument as in the first part of the proof of Lemma 1 it can be easily shown that F is maximized with $L_k \geq L^i_{fair}$. Thus,

$$max(F) = \max_{L_k \geq L^i_{fair}} \left(\sum_{k \in T} L^i_{fair} \frac{l_k}{L_k} \right) \tag{3.41}$$

Defining U as the set of flows of the user and I as the set of packets of flow i in the interval (t_1, t_n) we have

$$\frac{\sum_{i \in U} \sum_{k \in I} \frac{l_k}{L_k}}{\sum_{k \in T} \frac{l_k}{L_k}} = 1 \tag{3.42}$$

Let us define

$$W_i = \frac{\sum_{k \in I} \frac{l_k}{L_k}}{\sum_{k \in T} \frac{l_k}{L_k}} \tag{3.43}$$

Then according to Equation 3.42 we have

$$\sum_{i \in U} W_i = 1 \tag{3.44}$$

Grouping the sum of Equation 3.41 into flows

$$max(F) = \max_{L_k \geq L^i_{fair}} \left(\sum_{i \in U} L^i_{fair} \sum_{k \in I} \frac{l_k}{L_k} \right) \tag{3.45}$$

and multiplying and dividing by $\sum_{k \in T} \frac{l_k}{L_k}$ yields

$$max(F) = \left(\sum_{i \in U} L^i_{fair} \cdot W_i \right) \cdot \left(\max_{L_k \geq L^i_{fair}} \sum_{k \in T} \frac{l_k}{L_k} \right) \tag{3.46}$$

Since the condition $L_k \geq L^{min}_{fair}$ is less restrictive than $L_k \geq L^i_{fair}$,

$$\max_{L_k \geq L^i_{fair}} \left(\sum_{k \in T} \frac{l_k}{L_k} \right) \leq \max_{L_k \geq L^{min}_{fair}} \left(\sum_{k \in T} \frac{l_k}{L_k} \right) \tag{3.47}$$

and Equation 3.39 provides a bound for this term, we can give the following upper bound for the service received by the user in the interval (t_1, t_n)

$$\left(\sum_i L^i_{fair} \cdot W_i \right) \cdot \left(\frac{l_{max}}{L^{min}_{fair}} + K \left(2 + ln \frac{R}{L^{min}_{fair}} \right) + (t_n - t_1) \right) \tag{3.48}$$

Since $\left(\sum_i L^i_{fair} \cdot W_i \right) (t_n - t_1)$ represents the number of bits that the user is entitled to send during the interval $(t_n - t_1)$ (see Equation 3.13), the proof follows. ■

3.5 User Labeling and User Utility

The UFQ architecture is built around the assumption that users label their packets with $L_k = r_i^{send}/W_i$, where W_i is the weight of flow i in the user's utility function (*user labeling*, Equation 3.1). However, a user is allowed to label his packets with any label L_k with the only restriction of the ingress label control, which is much less restrictive on account of avoiding per-flow state at the ingress. A natural concern is whether a user can possibly benefit from labeling his packets with L_k different than r_i^{send}/W_i.

We cannot answer the above question with full generality, but we can analyze the same simplified situation as for Theorem 2 with the additional assumption that the sending rate of all flows is constant. Theorem 3 states that, in these conditions, a user sending n flows, all of them suffering from congestion, maximizes his utility when labeling his packets with $L_k = r_i^{send}/W_i$.

We conclude that, considering that all flows are susceptible to suffer from congestion, it is reasonable to assume that users label their packets with $L_k = r_i^{send}/W_i$, using an accurate estimation of flow i's sending rate r_i^{send} such as the one in Equation 3.2.

Theorem 3 *Consider a user sending n flows at a constant bit rate r_i^{send} through n bottleneck links, all links with a constant* fair label. *Then, the user maximizes his utility when labeling the packets of flow i with $L_k = r_i^{send}/W_i$, where W_i is flow i's weight in the user's utility function.*

Proof: We first use Theorem 2 to show the user's utility can be maximized with L_i constant (i.e. assigning the same label to all packets of flow i). Theorem 2 concludes that a user cannot obtain additional asymptotic throughput by labeling the packets of a flow with different labels. In addition, labeling flows with L_i constant gives enough flexibility to the user to freely divide among his flows the bandwidth allocated to him. As a consequence, the user's utility function can be maximized with L_i constant. In the following we take L_i constant.

Since the sending rates of all flows are constant, the restriction imposed by the ingress label control can be approximated by

$$\sum_{i \in U} \frac{r_i^{send}}{L_i} \leq 1 \tag{3.49}$$

where U is the set of flows of the user.

With the above, the problem of maximizing the user's utility can be expressed as the following optimization problem:

$$\begin{array}{rlr} \text{maximize} & \sum_{i\in U} W_i \cdot log(r_i) & (3.50) \\ \text{subject to} & \sum_{i\in U} \frac{r_i^{send}}{L_i} \leq 1 & (3.51) \\ \text{over} & L_i \geq 0 & (3.52) \end{array}$$

With the change of variable $\alpha_i = 1/L_i$, the objective function (3.50) is differentiable and stricty concave and the feasible region (3.51),(3.52) is compact; hence a maximizing value of α_i exists and can be found by Lagrangian methods. Consider the Lagrangian form:

$$L(\alpha_i, \lambda) = \sum_{i\in U} W_i \cdot log(r_i) + \lambda \left(\sum_{i\in U} r_i^{send} \cdot \alpha_i - 1 \right) \tag{3.53}$$

From Equations 3.8 and 3.9 we have

$$r_i = r_i^{send} \frac{L_{fair}^i}{L_i} = r_i^{send} \cdot L_{fair}^i \cdot \alpha_i \tag{3.54}$$

where L_{fair}^i is the *fair label* of flow i's bottleneck link.

Thus,

$$L = \sum_{i\in U} W_i \cdot log(r_i^{send} \cdot L_{fair}^i \cdot \alpha_i) + \lambda \left(\sum_{i\in U} r_i^{send} \cdot \alpha_i - 1 \right) \tag{3.55}$$

Then

$$\frac{\partial L}{\partial \alpha_i} = \frac{W_i}{\alpha_i} + \lambda \cdot r_i^{send} \tag{3.56}$$

Hence, at the maximum the following condition holds:

$$\frac{W_i}{\alpha_i} + \lambda \cdot r_i^{send} = 0 \tag{3.57}$$

The combination of Equations 3.57 and 3.51 yields

$$\lambda = -1 \tag{3.58}$$

As a consequence, the maximum utility is achieved by

$$\alpha_i = \frac{W_i}{r_i^{send}} \tag{3.59}$$

Finally, undoing the previous change of variable, we have that the utility of a user is maximized when he labels his packets with

$$L_k = \frac{r_i^{send}}{W_i} \tag{3.60}$$

which proofs the theorem. ■

3.6 Simulations

In this section we evaluate our algorithm by simulation. To provide some context, we compare UFQ's performance to three additional mechanisms for sharing resources: *FQ per-user*, *FQ per-flow* and TCP.

Fair Queuing (FQ) [26] is a queuing algorithm that aims at equally distributing the bandwidth of a link among traffic aggregates. In the *FQ per-user* approach, FQ is configured such that each traffic aggregate corresponds to the traffic generated by one user, in such a way that the link's bandwidth is divided equally among the users sending through this link. The *FQ per-user* approach is the basis of the *User Share Differentiation* (USD) architecture [39]. Note that USD, in contrast to UFQ, stores information for each user at core nodes, which results in a higher complexity.

In the *FQ per-flow* approach, FQ is configured such that each traffic aggregate corresponds to one flow, in such a way that the link's bandwidth is divided equally among the flows sending through the link. [33, 34, 35, 36] provide *FQ per-flow* without the need of storing per-flow state in core nodes.

The mechanism used for bandwidth sharing in the current Internet is the TCP protocol, which relies on the responsive behavior of the end-hosts to congestion. Active queue management schemes such as RED (Random Early Discarding) [40] aim at smoothening the behavior of TCP by providing early notification of congestion. Unless stated otherwise, simulation results for TCP will be provided using RED in the routers.

We have examined the behavior of UFQ under a variety of conditions, comparing its bandwidth allocations with the theoretical *user maxmin fair* (UMMF) distributions. Simulations 3.6.1 to 3.6.5 study the features of *user maxmin fairness* for bandwidth sharing and compares them with the other mechanisms. These simulations have been performed with constant bit rate UDP sources. Simulations 3.6.6 and 3.6.7 study some features of the UFQ mechanism. Finally, the support of different traffic models (TCP and ON-OFF sources) is analyzed in simulations 3.6.8 (one link) and 3.6.9 (several links).

All simulations have been performed in ns-2 [41]. Unless otherwise specified, we use the following parameters for the simulations. All the flows of a user have the same weight. Each output link has a capacity of 10 Mbps, a latency of 1 ms and a buffer of 64 KB. In the RED case, the first threshold is set to 16 KB and the second to 32 KB. The fair queuing (FQ) discipline is implemented with the weighted round-robin (WRR) scheduler. The packet size is set to 1000 bytes.

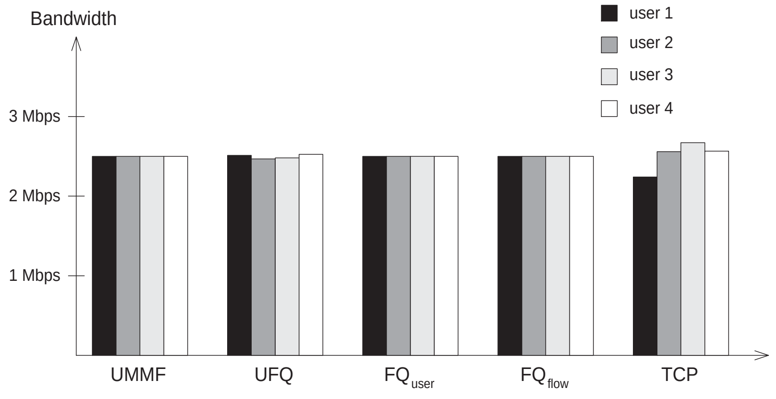

Figure 3.3: Single Flow - One Link.

3.6.1 Single Flow - One Link

Figure 3.3 shows the resulting bandwidth distribution with the various mechanisms when a 10 Mbps link is congested by four users sending a single flow each. It can be seen that the *user maxmin fairness* criterion (UMMF) distributes the link's bandwidth among the four users equally. The results provided by the UFQ mechanism are very close to the ideal results (UMMF). Note that in this simple scenario the other three approaches (*FQ per-user*, *FQ per-flow* and TCP) distribute the link's bandwidth similarly to UFQ.

3.6.2 Single Flow - Several Links

Figure 3.4 shows the bandwidth distribution when four users are sending one flow through a different number of equally congested links (see Figure 3.5). Also in this case, UFQ distributes the bandwidth such that the four users receive the same throughput.

FQ per-user and *FQ per-flow* distribute the bandwidth in the same way as UFQ. In contrast, TCP gives a better treatment to those flows with a lower number of hops.

3.6.3 Several Flows - One Link

Figure 3.6 shows the bandwidth distribution when one link is congested by four users transmitting each a different number of flows (user i transmits i flows). With UFQ all users receive the same throughput.

With *FQ per-user* the throughput distribution is the same as with UFQ. In contrast, *FQ per-flow* and TCP favor those users who are sending more flows, giving to each user a throughput proportional to his number of flows.

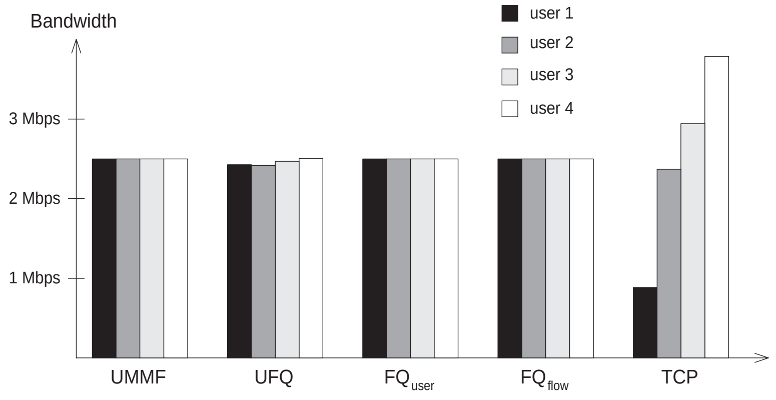

Figure 3.4: Single Flow - Serveral Link.

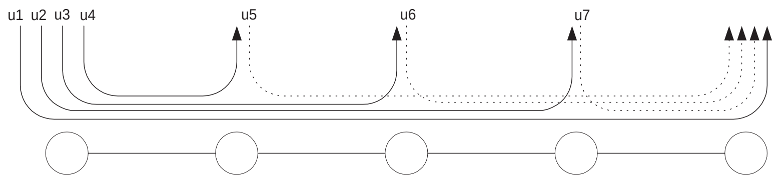

Figure 3.5: Simulation scenario, Single Flow - Several Links.

This is because *FQ per-flow* and TCP distribute the bandwidth on a per-flow basis.

3.6.4 Several Paths - Uniform Level of congestion

Figure 3.7 shows the bandwidth distribution for the case of users sending through different paths, when the level of congestion of all the links is the same (see Figure 3.8). UFQ provides all users with the same throughput.

In contrast to UFQ, the other approaches (*FQ per-user*, *FQ per-flow* and TCP) favor those users who are sending through more paths, giving them a throughput proportional to their number of paths. This is because these approaches distribute the bandwidth locally (either on a per-link basis – *FQ per-user*, on a per-flow basis - TCP, or both – *FQ per-flow*). The fact that UFQ works with overall network resources instead of locally is one of its key aspects, as compared to other existing approaches.

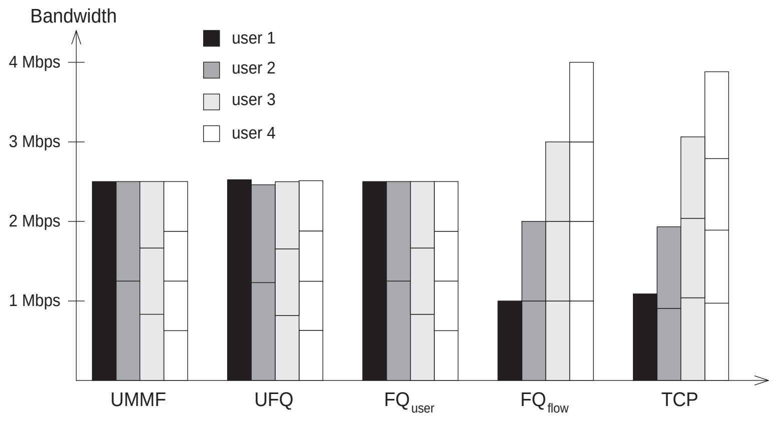

Figure 3.6: Several Flows - One Link.

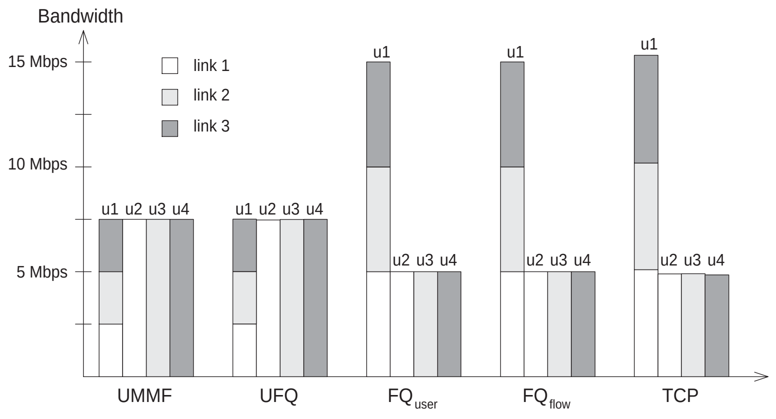

Figure 3.7: Several Paths - Uniform Level of congestion.

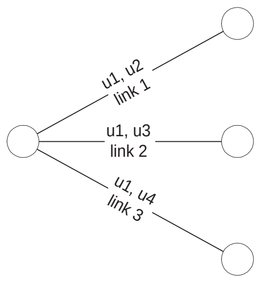

Figure 3.8: Simulation scenario, Several Paths - Uniform Level of congestion.

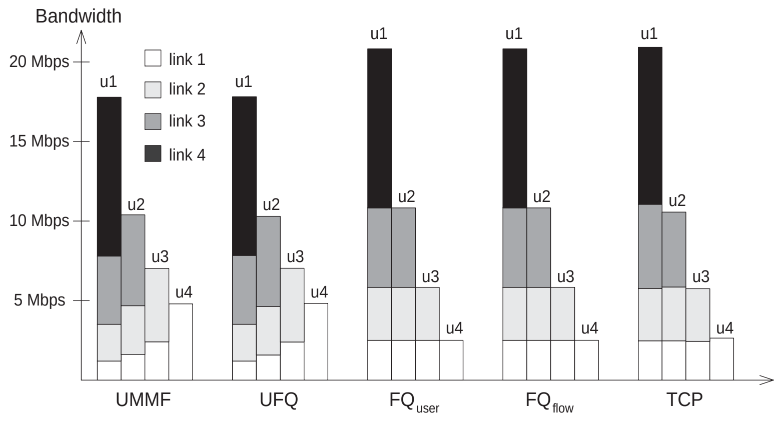

Figure 3.9: Several Paths - Heterogeneous Level of congestion.

3.6.5 Several Paths - Heterogeneous Level of congestion

Figure 3.9 shows the bandwidth distribution for the case of users sending through different paths, when the level of congestion of the links is variable (see Figure 3.10).

In the results of Figure 3.9 it can be seen that, with UFQ, user i receives a larger throughput than user $i + 1$. This is because user i is sending his flows through less congested links (in average) than user $i + 1$.

With the other approaches (*FQ per-user*, *FQ per-flow* and TCP) user i also receives a larger throughput than user $i + 1$. However, with these approaches the difference between the throughputs is larger. The reason is the same as for the previous simulation: while these three approaches distribute the bandwidth on a local basis, UFQ takes into account the overall network resources. For example, with *FQ per-user*, *FQ per-flow* and TCP, bandwidth in link 1 is distributed such that all users receive 2.5 Mbps. Instead, UFQ gives four times more bandwidth in this link to user 4 (4.8 Mbps), than to user 1 (1.2 Mbps), on account of the fact that user 4 is sending only through this link, while user 1 is sending to three additional links.

3.6.6 Intra-user Differentiation

In UFQ, a user expresses the relative value of his flows with the use of weights. In order to study this feature we repeated the experiment of simulation 3.6.3 assigning different weights to the two flows of user 2: $W_1 = 0.33$ and $W_2 = 0.66$ (note that $W_1 + W_2 = 1$).

Figure 3.11 plots the receiving rates averaged over 1 second interval for

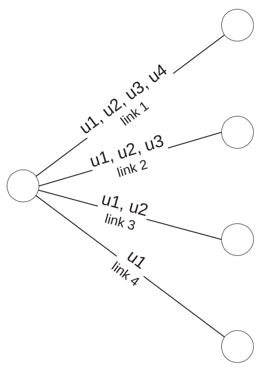

Figure 3.10: Simulation scenario, Several Paths - Heterogeneous Level of congestion.

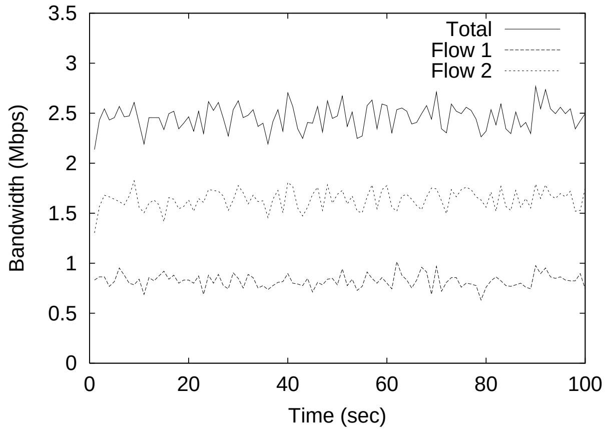

Figure 3.11: Intra-user differentiation.

flow 1, flow 2 and the total of user 2. It can be observed that the throughput received by flow 2 is twice as much as the received by flow 1, which matches the user's preferences.

3.6.7 Ingress label control

In order to assess the effectiveness of the ingress label control algorithm described in Section 3.2, we repeated the previous experiment but with the weights $W_1 = 3.33$ and $W_2 = 6.66$. Note that in this case $W_1 + W_2 = 10$, i.e. user 2 misbehaves and assigns lower labels than he should.

The results obtained from the above configuration are plotted in Figure 3.12. We can observe that the ingress control is effective since user 2 does not gain any excess service by misbehaving. In addition, the intra-user

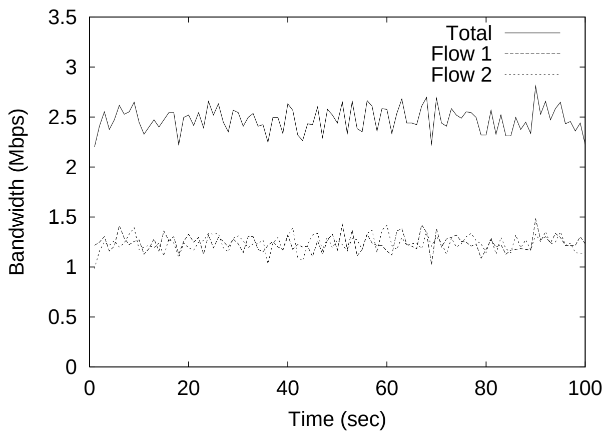

Figure 3.12: Ingress label control.

	TCP		Bursty UDP	CBR UDP
	sources	RTT	avg. rate	avg. rate
user 1	-	-	-	10 Mbps
user 2	-	-	-	15 Mbps
user 3	-	-	15 Mbps	-
user 4	10	20 ms	10 Mbps	-
user 5	5	20 ms	-	10 Mbps
user 6	5	20 ms	-	-
user 7	10	50 ms	-	-
user 8	10	20 ms	-	-

Table 3.1: Single Flow - One Link

differentiation feature is lost.

3.6.8 Different Traffic Models - One Link

So far we have only considered constant bit rate UDP traffic. We now study the behavior of UFQ under different traffic models. We consider a 40 Mbps link congested by 8 users sending a mixture of constant bit rate UDP, bursty UDP and endless TCP traffic. The bursty UDP traffic consists of an aggregate of ON/OFF sources, with ON periods following a Pareto distribution (average 50 ms), OFF periods exponentially distributed (average 50 ms) and a sending rate in the ON period of 2 Mbps. Table 3.1 shows the characteristics of the traffic sent by each user. For those users sending both TCP and UDP traffic, weights are set to equally distribute the user's bandwidth between TCP and UDP.

Figure 3.13 shows the resulting bandwidth distribution according to the *user maxmin fairness* criterion (UMMF), UFQ scheduling, a standard FIFO

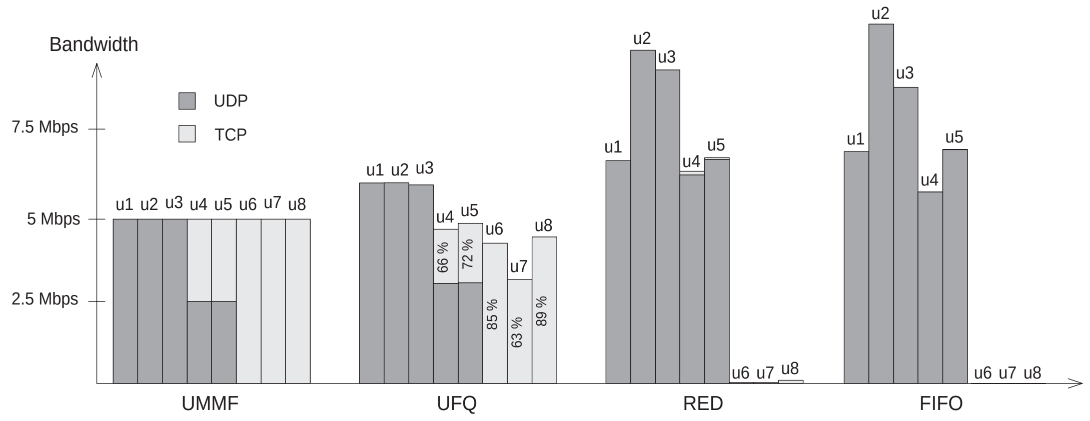

Figure 3.13: Different Traffic Models - One Link.

router and a RED router.

From these results we conclude that the bandwidth received by a user with the UFQ architecture depends on the level of responsiveness of his traffic. All users sending non-responsive UDP traffic receive the same throughput, independent of the sending rate and level of burstiness. In contrast, the throughput received by TCP traffic depends on the level of responsiveness of this traffic, determined by the number of TCP sources and their RTT[1]. However, from the results obtained it can be observed that TCP behaves reasonably well; in all cases TCP receives a throughput between 60% and 90% of its fair share rate, both when competing with UDP traffic of the same or a different user. Note that with FIFO and RED TCP traffic is starved.

The FBA-TCP architecture [42] has been proposed in the context of core stateless fair queuing for flows (specifically, within [33]) to improve the level of fairness between TCP and UDP. This is achieved by setting the maximum congestion window of TCP to the product of the RTT and L_{fair}. This approach could also be used within UFQ simply by adding the flow's weight W_i to this product.

In [24] we proposed a scheme based on control theory that improves the performance of TCP in the context of DiffServ networks by early marking some packets as *out*. This idea could be applied to UFQ by marking some packets with higher labels (note that high labels in UFQ is equivalent to *out*

[1]The RTT has two opposite effects: small RTTs give higher speeds for increasing the congestion window, which contributes to increase the throughput; at the same time, though, small RTTs increase the level of synchronization between TCP sources, which leads to a lower throughput. We have observed the same type of behavior for TCP in the context of DiffServ networks [22].

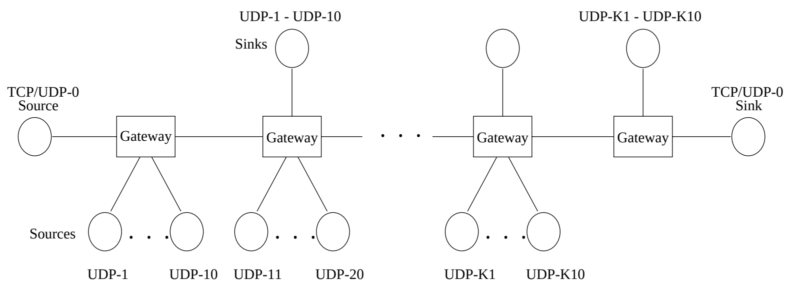

Figure 3.14: Simulation scenario, Different Traffic Models - Several Links.

in DiffServ). The advantage of this approach as compared to FBA-TCP is that it does not require to modify the TCP protocol.

3.6.9 Different Traffic Models - Several Links

We now analyze how the behavior of UFQ when a user is sending through more than one congested link. For this purpose we performed the experiment shown in Figure 3.14. In this experiment the target user is sending two flows, one TCP and one UDP (UDP-0) sending at its fair share rate. All links have a 10 Mbps capacity and are congested by 10 UDP flows (UDP-K1 to UDP-K10) sending at 1.5 Mbps each.

Figures 3.15 and 3.16 show the fraction of the flow's fair share rate received by the UDP and TCP flows of the target user. The behavior of the UDP flow is very close to the ideal: its throughput always keeps very close to its fair rate, independent of the number of congested links traversed. TCP performs reasonably well, even though its throughput decreases with the number of congested links: after traversing 5 congested links the throughput decreases from 70% to 60% of the source's fair share rate.

Note that with FIFO and RED the throughput of UDP decreases sharply with the number of congested links, while TCP is starved.

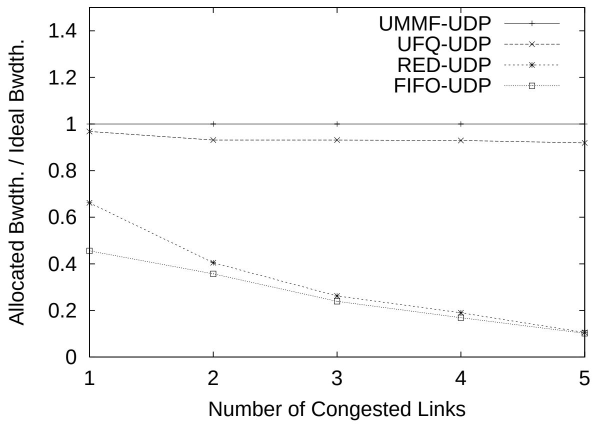

Figure 3.15: UDP throughput as a function of the number of congested links.

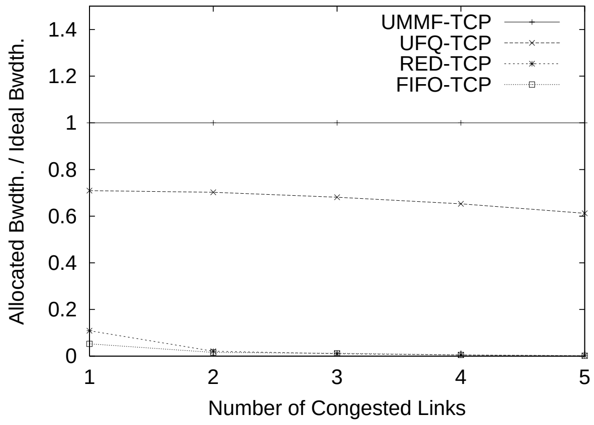

Figure 3.16: TCP throughput as a function of the number of congested links.

3.7 Summary and Discussion

In this chapter we have proposed the User Fair Queuing (UFQ) mechanism. This architecture is inspired by previous work on core stateless queuing.

In UFQ, a user is allowed to assign any label values to his packets to indicate their relative priority. At the ingress, an algorithm is used to control these labels assigned by the user. We have shown that the proposed label control does not allow the asymptotic throughput of a user to exceed its fair rate.

The bandwidth distribution resulting from UFQ depends on the way users label their packets. We have shown that, if users label their packets in order to maximize their level of satisfaction or utility, then the resulting bandwidth allocation is *user maxmin fair*.

The fact that neither admission control nor signaling are required strongly contributes to the simplicity of UFQ. The fact that no per-user state is kept at core nodes makes the proposed architecture scalable.

Chapter 4

User Fair Differentiation

In the previous chapter we dealt with the problem of distributing bandwidth equally among users. In this chapter we extend the problem to provide service differentiation, i.e. to allocate more bandwidth to those users that pay more.

Research on service differentiation is proceeding along two different directions: those proposals that use admission control and those that do not. In the approach with admission control [29], it is possible to control the amount of traffic allowed inside the network. In this way, traffic that would decrease the network service to a level lower than a desired limit can be stopped and an admitted user can be assured of his requested performance level. This approach, which we refer to as Absolute Service Differentiation, can be considered as trying to meet the same goals as IntServ, i.e. absolute performance levels, but pushing complexity admission control and traffic policing to the edge and to the Bandwidth Broker [43] and thus avoiding per-flow state in the core routers.

The second approach, which we refer to as Relative Service Differentiation, cannot prevent flooding of the network using admission control, and the only option to provide service differentiation is to forward packets in the network nodes with a quality according to their relative priority. Therefore, absolute performance levels are not guaranteed and only relative ones can be provided. The advantage of Relative Service Differentiation is that it is easier to deploy and manage (in Chapter 2 we have already mentioned the difficulties of admission control in DiffServ).

One of the models proposed for the Relative Service Differentiation is the Olympic Service Model [44]. The *User Fair Differentiation* (UFD) architecture proposed in this chapter extends the *User Fair Queuing* mechanism of Chapter 3 to provide service differentiation according to the Olympic Service Model.

4.1 The Olympic Service Model

In this section we describe the Olympic Service Model and discuss its validity and limitations.

The Olympic Service Model consists of three service classes: in order of increasing priority, *bronze*, *silver*, and *gold*. Packets are assigned to these classes according to the service contracted by their sender. Then, packets assigned to the gold class experience a better treatment than packets assigned to the silver class, and the same type of relationship exists between the silver and bronze classes. The Olympic Service Model must be strongly coupled with a pricing scheme that makes the gold class more costly than the silver class and the silver class more costly than the bronze class.

4.1.1 Olympic Service Model for Elastic Traffic

In the following, we justify the validity of the Olympic Service Model for Elastic Traffic. We will base the study on utility functions. As explained in Chapter 1, applications are divided into two groups: elastic applications and real-time applications. The utility function for elastic traffic has been discussed in Chapter 2; even though elastic applications benefit from an increasing amount of resources, they can still work with a low amount of network resources (see Figure 2.1 in Chapter 2).

In the Olympic Service Model the amount of network resources received by each user is not defined but depends on the level of congestion in the network. This uncertainty about the amount of network resources associated to a class is not particularly harmful in the case of elastic applications since, as said above, this kind of applications can still work with a low amount of network resources. Also, with the Olympic Service Model, the higher the priority of the class contracted by a user, the more network resources this user will receive. As a consequence, a higher priority class will always lead to a higher utility for elastic traffic, independent of the level of congestion in the network:

$$u_i = f(congestion);\ \forall\ congestion,\ p_i > p_j \Rightarrow u_i > u_j \tag{4.1}$$

where p_i is the class priority (i.e. bronze, silver or gold) and u_i is the utility experienced.

Since with the Olympic Service Model, elastic applications always experience a positive performance, and this performance increases with the class priority, we conclude that this model provides a valid service for elastic traffic. Note that, with this approach, admission control can be avoided while still providing a good service for this type of traffic.

Real-time applications, in contrast to the elastic ones, need a minimum amount of network resources to perform well, and perform badly with an amount of resources lower than this threshold. Examples of such applications are link emulation, audio and video. In Chapter 5 the requirements of real-time traffic are more thoroughly studied.

The uncertainty about the amount of network resources associated to a class in the Olympic Service Model may lead to a high priority class receiving a lower amount of resources than the minimum required and, consequently, experiencing a null performance. Also, with this model, an increase in the class priority will not always be beneficial: it will only be beneficial if it leads to an amount of network resources higher than the minimum required.

We conclude that the Olympic Service Model is not appropriate for real-time applications. This kind of applications would be better handled with admission control: without admission control, the level of congestion of the network cannot be controlled, and, as a consequence, utility cannot be guaranteed to applications that need a certain amount of resources to work properly. The Expedited Forwarding of IETF's DiffServ architecture [45] handles real-time traffic in this way. In Chapter 5 we provide a real-time traffic extension for the UFD architecture presented here. This extension relies on a user-based admission control scheme to guarantee that a real-time application receives the required minimum amount of resources.

4.1.2 Sender-based approach

The fact that the Olympic Service Model is sender-based imposes some limitations to its functionality. With a sender-based approach, a user can influence the treatment experienced by the packets he is sending, but not the treatment experienced by the packets he is receiving.

The services that fit best the nature of a sender-based approach such as the Olympic Service Model are the *one-to-one* and *one-to-any services for sending*. An example of the one-to-one case could be a VPN service that connects two networks of a company; if the gold service class is contracted for both networks, the VPN service will experience a good quality. An example of the one-to-any case could be a company that does its business on the web and is willing to pay an additional price to provide its users with a fast feel of its web site. If this company contracts the gold service class, its users will experience a good service quality.

While the *one-to-one service for sending* can be relatively easily provided by IETF's DiffServ architecture [20], the *one-to-any service for sending* is much more complex and it is still an ongoing effort [21].

A *one-to-one service for receiving* does not match the nature of the

Olympic Service Model, which is sender-based, but it still could be indirectly provided. An example of such a service could be a user that frequently accesses a specific video server to download movies [46]. With the Olympic Service Model, this user could contract a high quality service with the video server, which would in turn contract the gold service class with the network for the delivery of movies to this specific user. This would result in a good service quality experienced by the user when using this service. Note that in this case the money transaction consists of two steps: a first step from the user to the video server and a second step from the video server to the network operator.

Finally, the *any-to-one service for receiving* cannot be supported by the Olympic Service Model. An example of such a service could be a user willing to pay an extra price for high speed Internet access. Due to the usefulness of this service, the lack of support for it is an important limitation. The problem with any-to-one services is that they necessarily require some kind of signaling between the user and the ingress point of the packets received by him. Since the lack of signaling strongly contributes to the simplicity of an architecture, we conclude that the Olympic Service Model trades off functionality with simplicity. Note that within the IETF's DiffServ architecture, the support of any-to-one services has not been addressed yet.

4.1.3 Discussion

The Olympic Service Model is a pricing scheme that provides more network resources to those users who pay more. With this model users are not given absolute guarantees but relative ones: with the gold service class the quality experienced is better than with the silver service class, but this quality is left undefined and depends on the network conditions. Same kind of relationship exists between silver and bronze service classes.

The main advantage of the Olympic Service Model is its simplicity. The fact that the model is sender-based avoids the need for signaling, and its relative nature eliminates the need for admission control. However, this simplicity comes together with some limitations. With a sender-based approach, there are some services like Internet access that cannot be provided. The relative guarantees provided by the Olympic Service Model are well suited for elastic traffic, but not for real-time traffic, which requires of a more complex scheme with some kind of admission control providing absolute guarantees.

We conclude that the Olympic Service Model trades off functionality by simplicity, but still solves some major of the current Best Effort model. Given the current difficulties in the deployment of IETF's DiffServ and IntServ models, mainly due to their complexity, a pricing scheme like the Olympic Service

Model could find its application as the next step after the Best Effort model in the evolution of the Internet. In the following we propose an extension for elastic traffic of the UFQ architecture presented in Chapter 3 based on this model. In Chapter 5 we propose an extension for real-time traffic based on a different model.

4.2 The UFD architecture

The *User Fair Differentiation* (UFD) architecture extends the *User Fair Queuing* (UFQ) mechanism to implement the Olympic Service Model. This is done using the Proportional Differentiation criterion for bandwidth sharing.

4.2.1 Proportional Differentiation for Bandwidth

In the UFD architecture, each class of the Olympic Service Model (bronze, silver and gold) is mapped to a share, in such a way that the bandwidth experienced by a user is **proportional** to the share that he has contracted. The share associated with each class is chosen by the network operator and depends on the desired level of differentiation between classes; however, the gold class should be mapped to a higher share than the silver, and the silver to a higher than the bronze. This proportional differentiation for bandwidth sharing is expressed by the following equation:

$$\frac{r_g}{s_g} = \frac{r_s}{s_s} = \frac{r_b}{s_b} \tag{4.2}$$

where r_g is the bandwidth that a user would experience if he contracted the gold service class, r_s the bandwidth that he would experience with the silver class and r_b with the bronze, and s_g, s_s and s_b are the shares associated with each class ($s_g > s_s > s_b$).

In order to implement this proportional differentiation, the UFD architecture modifies the user labeling and the ingress label control of Chapter 3 as described in the following. With these modifications, the good features of UFQ for bandwidth sharing are preserved, while service differentiation is provided. The resulting algorithm is illustrated in Figure 4.1.

Let s_u be the share contracted by user u. Then, the packets of this user are labeled at the user network (*user labeling*) with:

$$L_k = \frac{r_i^{send}}{W_i \cdot s_u} \tag{4.3}$$

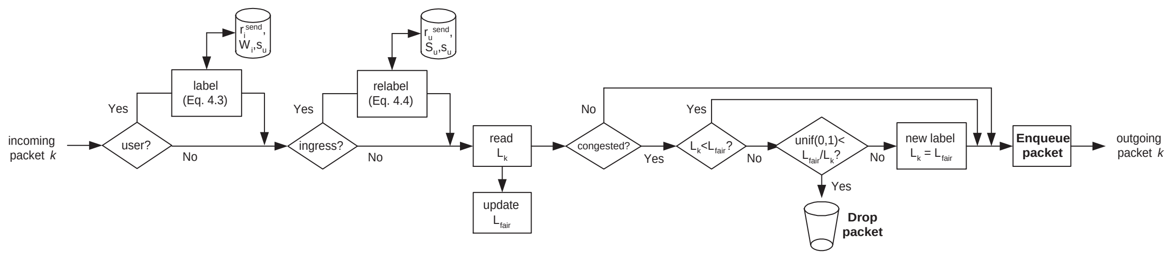

Figure 4.1: UFD algorithm.

and at the ingress they are relabeled (*ingress label control*) according to:

$$L_k^{new} = max\left(L_k, \frac{1}{s_u} \cdot \frac{(1-e^{-(\frac{l_k}{r_u^{send} \cdot K})})r_u^{send}}{1-e^{-(\frac{l_k}{r_u^{send} \cdot K})} \cdot (S_u)_{k-1}}\right) \tag{4.4}$$

The remaining functionality is left as described in Chapter 3. With the above, the rate experienced by flow i of user u is equal to:

$$r_i = s_u \cdot W_i \cdot L_{fair}^i \tag{4.5}$$

where L_{fair}^i is the fair label of flow i's bottleneck link.

The total rate experienced by user u, r_u, is:

$$r_u = \sum_{i \in U} r_i = \left(\sum_{i \in U} W_i \cdot L_{fair}^i\right) s_u \tag{4.6}$$

From the above equation, it can be seen that, with the approximation that L_{fair}^i is held constant, the rate received by a user u increases linearly with the contracted share s_u. We conclude that the bandwidth experienced by a user is proportional to the share of the class contracted by this user (s_g, s_s or s_b), which is the objective that we stated for the UFD architecture in Equation 4.2.

4.2.2 Inter-domain

Since the Internet consists of different domains, in order to provide end-to-end QoS, domains are required to cooperate. Thus, the QoS behavior when crossing domains (inter-domain part) is an essential aspect of any service differentiation architecture. In the UFD architecture, the inter-domain part is, as the intra-domain, based on shares; but in this case it is not a user who contracts a share to his domain (where the share contracted by a user is a function of his service class, i.e. bronze, silver or gold), but it is a domain that contracts a share with a neighboring domain. This share that one domain contracts with another will be divided among all the users sending from the first domain to the second. So, for example, if a domain has 100 users sending to another domain, and wants them to experience the same quality as one user of the other domain with a share of 1 contracted within that domain, then the first domain will have to contract a share of 100 to the second domain.

When crossing domains, for scalability reasons users have to be somehow aggregated. As it has been explained in the intra-domain part of the architecture, per-user state needs to be maintained at the edges in order to measure each user's rate. If users are not aggregated, boundary routers also need to keep state for every user crossing the router. The number of users crossing edge routers will always be relatively small, but this does not have to hold true for boundary routers between domains. Therefore, if users are not aggregated, boundary routers may need to keep a very large state and will then become bottlenecks concerning scalability.

The design goals for the inter-domain part of the UFD architecture are the following:

1. Neither per-user nor per-flow state, but only per-domain state should be kept in the boundary routers between domains.

2. The desired level of differentiation between the users of a domain (expressed by the shares contracted by these users) should be preserved when crossing domains.

3. The aggregated amount of resources received by the users of a first domain in a second domain should correspond to the share contracted by the first domain to the second.

In order to preserve the desired level of differentiation when crossing domains, the packets in the new domain should preserve the ratios between the labels they had in the old domain. This condition is expressed in the following equation:

$$\frac{L_1^{new}}{L_1^{old}} = \frac{L_2^{new}}{L_2^{old}} = \ldots = \frac{L_k^{new}}{L_k^{old}} = \beta \tag{4.7}$$

where L_k^{new} is the label of packet k in the new domain and L_k^{old} is its label in the old domain.

To ensure that the first domain is not labeling its packets with more resources than the amount contracted (condition 3), label control is performed at the ingress of the second domain (Equation 4.4 with $s_u = s_d$, where s_d is the share contracted by the first domain to the second). The combination of this with condition 2 leads to relabeling packets at the egress of a domain such that: *a)* the ratios between labels is preserved, and *b)* packets are not relabeled by the label control.

Condition b is satisfied by enforcing the following:

$$\frac{avg\left(\frac{l_k}{L_k^{new}}\right)}{avg\left(l_k\right)} r_d^{send} = s_d \tag{4.8}$$

where r_d^{send} is the sending rate from the first domain to the second.

Combining the above equation with Equation 4.7 (condition a) leads to the following expression for β:

$$\beta = \frac{r_d^{send}}{s_d} \cdot \frac{avg\left(\frac{l_k}{L_k^{old}}\right)}{avg\left(l_k\right)} \tag{4.9}$$

For the estimation of the above expression we use the following exponential averaging formula. Let t_k and l_k be the arrival time and length of the k^{th} packet from the first domain to the second. The estimation of β is updated for every new packet k:

$$\beta_k = (1 - e^{-(T_k/K)}) \frac{l_k}{T_k \cdot L_k^{old} \cdot s_d} + e^{-(T_k/K)} \cdot \beta_{k-1} \tag{4.10}$$

where $T_k = t_k - t_{k-1}$ and K is a constant (as in Chapter 3, we set $K = 100ms$).

The above value of β_k is then used to relabel packet k at the egress of the first domain with the following label value:

$$L_k^{new} = \beta_k \cdot L_k^{old} \tag{4.11}$$

Figure 4.2 illustrates the UFD algorithm for inter-domain communication. This algorithm is an addition to the UFD intra-domain algorithm of Figure 4.1.

Note that the proposed algorithm does not require to keep neither per-user nor per-flow state at the boundary routers between domains, which satisfies the remaining design goal (condition 1). As we argued above, this condition is necessary in order to provide scalable inter-domain communication (see Figure 4.3).

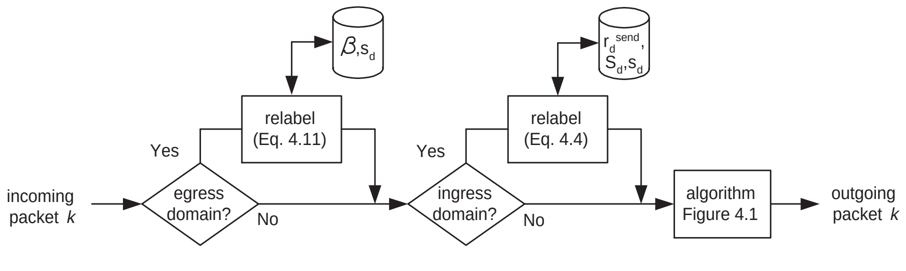

Figure 4.2: UFD inter-domain algorithm.

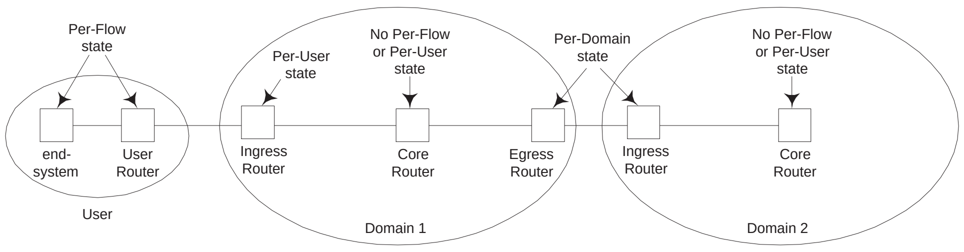

Figure 4.3: UFD architecture.

4.3 Comparison with existing approaches

In this section, we compare via simulation our architecture with the existing approaches also based on the Olympic Service Model. This comparison is done according to the two following targets:

Isolation In order to avoid that non-adaptive UDP sources can gain greater shares of network bandwidth while starving other, well-behaved, TCP sources, some form of traffic isolation inside the network is needed.

Differentiation According to the design principle of the Olympic Service Model, the users who have contracted the gold class should experience a larger bandwidth than the users who have contracted the silver class, and the same type of relationship between the silver and bronze classes.

The purpose of the simulations is to show the validity of the conceptual approach of the UFD architecture for isolation and differentiation as compared to the other Olympic Service Model architectures.

For this purpose, we simulated a simple scenario with three users sending through one bottleneck link of 10 Mbps: user 1 with a share of 3 (gold class), user 2 with a share of 2 (silver class) and user 3 with a share of 1 (bronze class). This scenario reflects the level of isolation and differentiation achieved by users belonging to different service classes.

We also simulated another scenario with a variable number of users for each service class: three users for the gold class, two for the silver class and one for the bronze class. This second scenario reflects the level of isolation between the different users of the same class, and the impact of the load in a class.

Finally, we simulated a scenario with a two-domain network topology with a bottleneck link of 10 Mbps in the second domain, in order to study the level of isolation and differentiation provided by the different architectures when crossing domains.

For the above scenarios we used both UDP CBR sources sending at 5 Mbps and endless TCP sources, hereafter referred as UDP and TCP respectively.

4.3.1 UFD

Tables 4.1 and 4.2 show the level of isolation and differentiation provided by the UFD architecture for both the one-user-per-class and several-users-per-class cases. The bandwidth experienced by each user is given in Kbps; in parenthesis this bandwidth is expressed as a percentage with respect to the user's fair share of bandwidth.

Simulation results show that, when all flows are UDP (tests 3 and 6), UFD provides the desired level of differentiation independent of the number of users per class. However, when the packet drops of the UFD architecture interact with the congestion control of TCP, results deviate from the desired ones. Nevertheless, the resulting bandwidth allocations are still fairly good; in all cases, users obtain more than 75% of their fair share of bandwidth.

Note that the solutions to improve TCP performance in UFQ explained in Section 3.6.8 ([42] and [24]) could also be applied to improve TCP performance in UFD.

To test the inter-domain part of the UFD architecture we simulated a scenario in which the users of Table 4.2 are sending their flows to a second domain with which the first domain has contracted a share of 5, and in this second domain they are competing with a user that has contracted a share

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4311 (86%)	UDP	4969 (99%)	UDP	4917 (98%)
2	2	TCP	3399 (102%)	TCP	3006 (90%)	UDP	3389 (102%)
3	1	TCP	2253 (135%)	UDP	1979 (119%)	UDP	1693 (102%)

Table 4.1: UFD. One user per class

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	2200 (103%)	UDP	2351 (110%)	UDP	2154 (101%)
2	3	TCP	2051 (96%)	UDP	2317 (108%)	UDP	2155 (101%)
3	3	TCP	2029 (95%)	TCP	1755 (82%)	UDP	2160 (101%)
4	2	TCP	1342 (94%)	UDP	1583 (111%)	UDP	1437 (101%)
5	2	TCP	1551 (109%)	TCP	1166 (82%)	UDP	1405 (98%)
6	1	TCP	790 (110%)	UDP	781 (109%)	UDP	683 (96%)

Table 4.2: UFD. Several users per class

		TEST 7		TEST 8		TEST 9	
user	share	source	Kbps	source	Kbps	source	Kbps
1-d1	3	TCP	1688 (94%)	UDP	1935 (108%)	UDP	1782 (100%)
2-d1	3	TCP	1606 (90%)	UDP	1965 (110%)	UDP	1778 (99%)
3-d1	3	TCP	1723 (96%)	TCP	1394 (78%)	UDP	1794 (100%)
4-d1	2	TCP	1187 (100%)	UDP	1293 (109%)	UDP	1190 (100%)
5-d1	2	TCP	1237 (104%)	TCP	944 (79%)	UDP	1190 (100%)
6-d1	1	TCP	680 (114%)	UDP	661 (111%)	UDP	602 (100%)
d1	5	TCP	8121 (97%)	TCP-UDP	8192 (98%)	UDP	8336 (100%)
7-d2	1	TCP	1657 (99%)	UDP	1812 (109%)	UDP	1663 (100%)

Table 4.3: UFD. Inter-domain

of 1 with the second domain. In the simulation results of Table 4.3 we can see that the differentiation between the users of the first domain is preserved in the second domain, and in total these users get approximately the share that their domain has contracted with the second domain. These were the objectives of the inter-domain architecture explained in Section 4.2.2.

4.3.2 User Share Differentiation (USD)

The User Share Differentiation (USD) architecture [39] also maps each service class into a share. The share corresponding to each user is stored by the core nodes of the network. The core nodes use this share to give the packets of the corresponding user their fair part of the forwarding capacity (using e.g. a WFQ scheduler). Note that since USD stores information for each user at core nodes, it has the problem of poorly scaling with respect to the number of users, and might result in implementation problems when applied to core routers in large domains.

The results for the intra-domain simulations (Tables 4.4 and 4.5) show that USD provides the required isolation and ensures the necessary differentiation according to the service classes. This perfect isolation can be achieved because in USD, in contrast to UFD, per-user state is stored at core routers.

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4999	UDP	4994	UDP	4996
2	2	TCP	3333	TCP	3336	UDP	3336
3	1	TCP	1666	UDP	1668	UDP	1667

Table 4.4: USD. One user per class

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	2142	UDP	2142	UDP	2142
2	3	TCP	2142	UDP	2142	UDP	2142
3	3	TCP	2142	TCP	2142	UDP	2142
4	2	TCP	1428	UDP	1428	UDP	1428
5	2	TCP	1428	TCP	1428	UDP	1428
6	1	TCP	714	UDP	714	UDP	714

Table 4.5: USD. Several users per class

For the inter-domain simulations of the USD architecture, we implemented user aggregation as suggested in [39], i.e., in the second domain, the users from the first domain are aggregated in classes with identical shares. In this case, users 1, 2 and 3 (gold class) are aggregated in a class in domain 2 with share 3, users 4 and 5 (silver class) in a class with share 2 and user 6 (bronze class) in a class with share 1. Note that since in USD users are statically assigned to a class, such a situation in which one class is more crowded than another can easily occur. The simulation results of Table 4.6 show that USD fails to guarantee proper differentiation due to user aggregation when crossing domains: all users receive the same treatment, independent of their service class. In addition, the inter-domain part of USD also fails to provide proper isolation, again due to aggregation effects: in test 8 we can see that TCP flows starve when sharing a class with UDP flows in the second domain.

		TEST 7		TEST 8		TEST 9	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	1672	UDP	2485	UDP	1667
2	3	TCP	1672	UDP	2504	UDP	1669
3	3	TCP	1655	TCP	3	UDP	1662
4	2	TCP	1665	UDP	3248	UDP	1666
5	2	TCP	1667	TCP	48	UDP	1666
6	1	TCP	1666	UDP	1666	UDP	1666

Table 4.6: USD. Inter-domain

4.3.3 SIMA

The Simple Integrated Media Access (SIMA) architecture [47, 48] defines different levels of service based on the Nominal Bit Rate contracted by the user. SIMA provides two types of services, one for real-time traffic and the other for non-real-time. Since the focus of this chapter is on elastic traffic, we will only consider the latter.

In SIMA, the sending rate of user u, r_u^{send}, is estimated at the ingress of the domain, and, based on this estimation, packets are labeled according to the following formula[1]:

$$L_k = \begin{cases} 6 & if x \geq 6 \\ int(x) & if 0 < x < 6 \\ 0 & if x \leq 0 \end{cases} \tag{4.12}$$

where x is

$$x = 4.5 - \frac{ln(r_u^{send}/NBR)}{ln(2)} \tag{4.13}$$

being $int(x)$ the integer part of x and NBR the user's Nominal Bit Rate.

Then, in case of congestion, packets are dropped at core nodes depending on their label L_k. The dropping is performed using the Weighted Random Early Detection (WRED) [49] active queue management algorithm, which is configured with seven separate thresholds (one for each label value).

In the simulations of SIMA, we assigned a NBR of 3 Mbps to the users of the gold service class, 2 Mbps to the users of the silver service class and 1 Mbps to the bronze. Simulation results for the intra-domain case are shown in Tables 4.7 and 4.8. These results show that SIMA provides a good level of differentiation in the case of responsive TCP sources (tests 1 and 4), even though the throughputs obtained are not proportional to the NBRs. When dealing with non-responsive UDP sources, however, this feature is lost. In tests 2, 3, 5 and 6, we can see that the bronze service class starves when competing with the gold and silver service classes. This is due to the fact that packet drops in SIMA are not probabilistic but based on thresholds: a user sending too much, thus, will see all his packets dropped. Also in tests 2, 3, 5, and 6, the UDP silver sources receive the same treatment as the gold ones. This is due to the coarse granularity of SIMA, which comes from the small number of label values used: a user sending at 5 Mbps receives the same label both for a NBR of 2 and 3 Mbps, since

[1]Note that since L_k can only take 7 different values, three bits are enough to store its value.

		TEST 1		TEST 2		TEST 3	
user	NBR	source	Kbps	source	Kbps	source	Kbps
1	3 Mbps	TCP	5325	UDP	4977	UDP	4944
2	2 Mbps	TCP	3063	TCP	4721	UDP	4922
3	1 Mbps	TCP	1572	UDP	287	UDP	132

Table 4.7: SIMA. One user per class

		TEST 4		TEST 5		TEST 6	
user	NBR	source	Kbps	source	Kbps	source	Kbps
1	3 Mbps	TCP	2330	UDP	2579	UDP	1991
2	3 Mbps	TCP	2331	UDP	2561	UDP	1978
3	3 Mbps	TCP	2357	TCP	1206	UDP	1985
4	2 Mbps	TCP	1140	UDP	2567	UDP	2015
5	2 Mbps	TCP	1128	TCP	1076	UDP	1995
6	1 Mbps	TCP	693	UDP	0	UDP	34

Table 4.8: SIMA. Several users per class

$$int\left(4.5 - \frac{ln(5/2)}{ln(2)}\right) = int\left(4.5 - \frac{ln(5/3)}{ln(2)}\right) = 3 \tag{4.14}$$

Finally, in test 5 it can be observed that the level of isolation provided by SIMA between UDP and TCP is not perfect but reasonable.

For the inter-domain simulations, we simulated a scenario in which the users of Table 4.8 are sending their flows to a second domain with which the first domain has contracted a NBR of 5 Mbps, and in this second domain they are competing with a user that has a NBR of 1 Mbps. In the simulation results of Table 4.9 we can see that SIMA does not provide proper differentiation when crossing domains: in tests 7 and 9 all users receive the same treatment, independent of their NBR in the first domain. In test 8 it can be

		TEST 7		TEST 8		TEST 9	
user	NBR	source	Kbps	source	Kbps	source	Kbps
1-d1	3 Mbps	TCP	1420	UDP	1991	UDP	821
2-d1	3 Mbps	TCP	1596	UDP	2014	UDP	848
3-d1	3 Mbps	TCP	2023	TCP	1	UDP	819
4-d1	2 Mbps	TCP	818	UDP	1995	UDP	837
5-d1	2 Mbps	TCP	1622	TCP	0	UDP	826
6-d1	1 Mbps	TCP	1238	UDP	2006	UDP	834
d1	5 Mbps	TCP	8717	TCP-UDP	8007	UDP	4985
7-d2	1 Mbps	TCP	1210	UDP	1991	UDP	5011

Table 4.9: SIMA. Inter-domain

seen that the inter-domain part of SIMA does not provide proper isolation either, since the TCP flows starve. This inter-domain behavior of SIMA is caused by the fact that the users from the first domain are treated like an aggregate in the second domain.

4.3.4 Delay Differentiation (DD)

In [50, 51, 52] different schedulers for proportional differentiation in delay are proposed. These schedulers basically schedule packets in such a way that the waiting time in the queue is inversely proportional to the share assigned to the corresponding service class.

In order to better understand the performance of this type of schedulers we ran the simulations corresponding to Tables 4.10 and 4.11 using the scheduler implementation of [53]. We can observe from the simulation results that delay differentiation provides a good level of differentiation for TCP traffic alone (tests 1 and 4), since the throughput obtained by a TCP flow is inversely proportional to its round-trip delay. For UDP traffic alone (tests 3 and 6), no differentiation in throughput is obtained; however, for this kind of traffic the delay alone may suffice to provide meaningful differentiation[2]. The main drawback of the delay differentiation approach, however, is expressed by the results of tests 2 and 5. We can observe in these tests that TCP sources almost starve when competing with UDP, which shows that the queuing delay as differentiation parameter cannot provide proper isolation.

We conclude that, even though delay can be an important differentiation parameter for delay sensitive (i.e. real-time) applications, it needs to be combined with bandwidth differentiation in order to provide proper isolation. This is the approach we have taken in the real-time extension of the UFD architecture that we have proposed in Chapter 5.

Inter-domain simulation results for the Delay Differentiation approach have not been provided since they do not differ from the intra-domain ones.

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4884	UDP	4784	UDP	3266
2	2	TCP	3370	TCP	398	UDP	3360
3	1	TCP	1745	UDP	4842	UDP	3376

Table 4.10: DD. One user per class

[2]Actually, delay is not very relevant for elastic traffic. It is relevant for real-time traffic, but, as we have argued in Section 4.1.1, the Olympic Service Model does not fit well the requirements of this traffic type.

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	2125	UDP	2481	UDP	1708
2	3	TCP	2132	UDP	2489	UDP	1643
3	3	TCP	2130	TCP	84	UDP	1676
4	2	TCP	1444	UDP	2455	UDP	1679
5	2	TCP	1444	TCP	20	UDP	1652
6	1	TCP	733	UDP	2469	UDP	1640

Table 4.11: DD. Several users per class

4.3.5 Class-Based Allocation (CBA)

A Class-based allocation approach, such as the one in [44], assigns a specific capacity to each service class. Each network flow that belongs to a certain class, therefore, shares a common set of resources with other flows in that class. This approach has the drawback that the service quality associated with a class is undefined, since it depends on the arriving load in that class: as the traffic in the Internet is extremely bursty [54], the load in each class and consequently its service quality fluctuates.

This behavior is shown in the simulation results of Tables 4.12 and 4.13. These simulations have been performed with the Weighted Round Robin implementation of [55], with a different queue for each class with a weight equal to the share of the class (3 for the gold class, 2 for the silver and 1 for the bronze). Table 4.12 shows that when the load in each class is uniform, the level of differentiation obtained is the desired. However, when the load in the higher priority classes is larger, the differentiation feature is lost: in the example of Table 4.13 we can see that all users get the same throughput, independent of the service class they have contracted (bronze, silver or gold). In addition, isolation is not provided inside each class, so the most aggressive sources eat up all the available bandwidth in a class (see test 5 in Table 4.13).

Inter-domain simulation results have not been provided for this approach either, since also in this case they do not differ from the intra-domain ones.

		TEST 1		TEST 2		TEST 3	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	4999	UDP	4994	UDP	4996
2	2	TCP	3333	TCP	3336	UDP	3336
3	1	TCP	1666	UDP	1668	UDP	1667

Table 4.12: CBA. One user per class

		TEST 4		TEST 5		TEST 6	
user	share	source	Kbps	source	Kbps	source	Kbps
1	3	TCP	1666	UDP	2496	UDP	1666
2	3	TCP	1665	UDP	2498	UDP	1670
3	3	TCP	1664	TCP	2	UDP	1663
4	2	TCP	1665	UDP	3284	UDP	1670
5	2	TCP	1667	TCP	48	UDP	1662
6	1	TCP	1666	UDP	1666	UDP	1666

Table 4.13: CBA. Several users per class

4.4 Summary and Discussion

The User Fair Differentiation (UFD) architecture extends the UFQ mechanism to provide service differentiation in a multi-domain environment. Service differentiation is based on the Olympic Service Model. We have shown that this model provides a valid service for elastic traffic but not for real-time.

The UFD approach to the Olympic Service Model is based on the Proportional Differentiation criterion for bandwidth. This criterion allocates a fixed number of times more bandwidth to a user of the gold service class than to a user of the silver service class, and the same for the silver and bronze service classes. This level of differentiation between service classes is preserved when crossing domains. The simulation results presented in this section show that UFD is effective in achieving these objectives.

Simulation results have also validated the isolation and differentiation features of the UFD architecture. These features have been compared, also via simulation, with other architectures that also implement the Olympic Service Model, namely, User Share Differentiation (USD), Simple Integrated Media Access (SIMA), Delay Differentiation (DD) and Class-Based Allocation (CBA). From the results obtained, we conclude that the UFD architecture is the only one that provides the isolation and differentiation features for both the intra-domain and the inter-domain cases.

Chapter 5

Extension for Real-Time Traffic

The applicability of the UFD architecture presented in the previous chapter is limited to elastic traffic. The main relevant parameter for this traffic type is bandwidth. In this chapter we extend the UFD architecture to support real-time traffic. We study the delay and bandwidth requirements of real-time applications and propose the *Real-Time User Fair Differentiation* (RT-UFD) extension to satisfy these requirements. The proposed extension has two aspects:

Delay Requirements The *Step Differentiation model for delay* we propose satisfies the delay requirements of real-time traffic. In UFD-RT, this model is implemented with the *traffic type separation* mechanism.

Bandwidth Requirements In order to satisfy the bandwidth requirements of real-time traffic we propose the *User-based Admission Control* (UBAC) scheme.

5.1 Step Differentiation for Delay

While the delay has a small impact on elastic applications, it is of key importance for the performance of real-time applications. In real-time transmissions, the source takes some signal, packetizes it, and then transmits it over the network. The network inevitably introduces some variation in the delay of each delivered packet. This variation has traditionally been called *jitter*. The receiver depacketizes the packet and the attempts to faithfully play back the signal. This is done by buffering the incoming data to remove the network induced jitter and then replaying the signal at some designated *play-back* point. Any data that arrives before its associated *play-back* point can be used to reconstruct the signal; data arriving after the *play-back* point

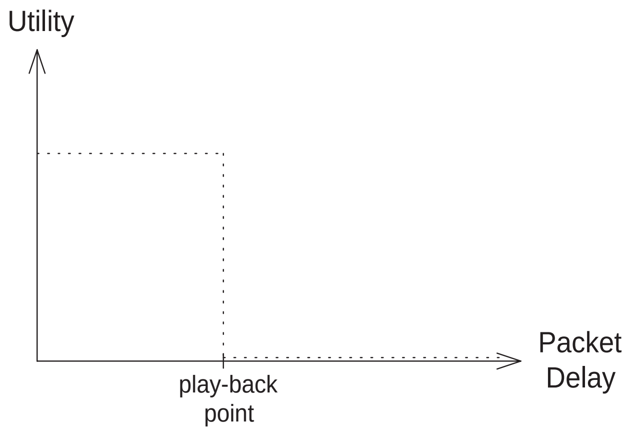

Figure 5.1: Utility function of real-time applications as a function of the delay.

is useless in reconstructing the real-time signal. This dependency of the performance of real-time applications on packet delay is illustrated in the utility function of Figure 5.1.

For delay differentiation in the RT-UFD extension, we have chosen the *Step Differentiation model*, in contrast to the Proportional Differentiation model for bandwidth of Section 4.2.1. The Step Differentiation model states that delay should be such that real-time packets either suffer a very low delay or an infinite delay (i.e. get dropped) depending on the share that has been assigned to their user. So, if D_k is the delay experienced by a packet k of user u, and s_u is the share contracted by user u, then the step differentiation imposes:

$$D_k = \begin{cases} \textit{very low} & s_u \textit{ high enough} \\ \infty & \textit{otherwise} \end{cases} \tag{5.1}$$

The choice of the step differentiation model for delay is justified by the utility function of Figure 5.1: since for real-time application packets have to arrive within a given delay bound (the *play-back* point) or otherwise they provide no utility to the user, only the amount of real-time traffic that can be transmitted by the network with a delay lower than the delay bound is accepted, and the rest of the real-time traffic is discarded.

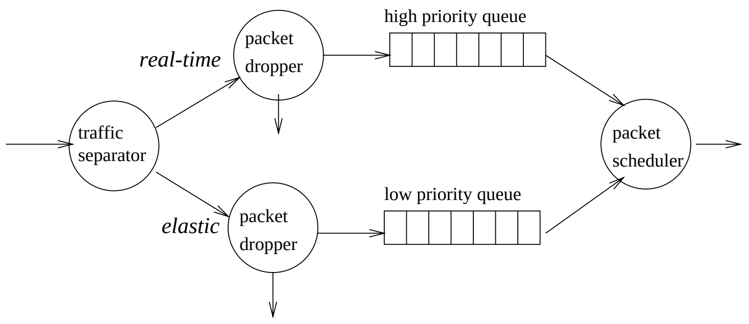

Figure 5.2: Traffic Type Separation.

5.2 Traffic Type Separation

In this section, we propose the traffic type separation mechanism to implement the Step Differentiation model for delay. This mechanism is inspired by the Two-Bit Architecture [56] in that we have one bit in the packet header indicating whether the packet belongs to a real-time or an elastic application, and for each link we have two queues: a high priority queue for real-time traffic and a low priority queue for elastic traffic.

The incoming packets are separated according to their type, i.e. elastic or real-time, at the traffic separator. The traffic separator then inputs the packet to the corresponding packet dropper. This mechanism is depicted in Figure 5.2.

For both traffic types, a *fair label* is calculated, i.e. L^{el}_{fair} for elastic traffic and L^{rt}_{fair} for real-time traffic. Elastic packets are allocated to the low priority queue if their label L_k is low enough ($L_k \leq L^{el}_{fair}$). Otherwise they are dropped with probability $d_k = 1 - L^{el}_{fair}/L_k$. Note that this is the dropping mechanism described in Section 3.3.

The same mechanism for dropping packets is used for real-time packets: they are allocated to the high priority queue if their label L_k is low enough (i.e., $L_k \leq L^{rt}_{fair}$) and they are dropped with probability $d_k = 1 - L^{rt}_{fair}/L_k$ otherwise.

With the queuing mechanism presented, elastic and real-time traffic are separated in such a way that the capacity of the link, C, is divided among them. A key point of RT-UFD is to choose the right proportion of the available capacity to be assigned to each traffic type. This assignment is enforced by means of the packet droppers (i.e. by adjusting properly the values of the fair labels for real-time and elastic traffic, L^{rt}_{fair} and L^{el}_{fair}).

If too much capacity is assigned to real-time traffic, the packet dropper of the real-time traffic will not be aggressive enough and we run the risk of accepting bursts of real-time traffic that fill up the real-time queue. This can result in real-time packets missing their delay bound due to queuing delay. Consequently, the part of the link's capacity used by real-time traffic needs to be limited to a maximum value, $\frac{1}{k} \cdot C$, to ensure small forwarding delays. $\frac{1}{k}$ is a constant that has to be chosen by the network operator as a function of the amount of real-time traffic that has to be supported by the network and the desired service quality for this real-time traffic.

In order to limit real-time traffic to $\frac{1}{k} \cdot C$, L^{rt}_{fair} is computed according to the following algorithm:

```
Every $K$ time units:
**if** $A^{rt} \geq C/k$ **then**
    $(L^{rt}_{fair})_{new} = (L^{rt}_{fair})_{old} \cdot \frac{C/k}{F^{rt}}$
**else**
    $(L^{rt}_{fair})_{new} =$ largest $L^{rt}_k$ observed
**end if**
```

where A^{rt} is the estimated aggregated arrival rate for real-time traffic, and F^{rt} is the estimated rate of the accepted traffic for real-time traffic.

Elastic traffic gets assigned the remaining capacity of the link, $C - F^{rt}$. Note that F^{rt} will at maximum equal $\frac{1}{k} \cdot C$; therefore elastic traffic is never starved. Also, elastic traffic will be allowed to use up to the full capacity of the link when there is no higher-priority real-time traffic. This is achieved by computing the fair label for elastic traffic, L^{el}_{fair}, according to the following algorithm:

```
Every $K$ time units:
**if** $A^{el} \geq C - F_{rt}$ **then**
    $(L^{el}_{fair})_{new} = (L^{el}_{fair})_{old} \cdot \frac{C-F^{rt}}{F^{el}}$
**else**
    $(L^{el}_{fair})_{new} =$ largest $L^{el}_k$ observed
**end if**
```

where A^{el} is the estimated aggregated arrival rate for elastic traffic, and F^{el} is the estimated rate of the accepted traffic for elastic traffic.

Delay differentiation is provided by means of the two level simple priority queuing shown in Figure 5.2. Whenever there are packets in the high priority queue, they will be forwarded by the packet scheduler first, and whenever there are no high priority packets to be forwarded, packets are forwarded from the low priority queue for elastic traffic. By assigning real-time packets to the high priority queue, we achieve a very low delay for those real-time packets that are not dropped, as stated by the Step Differentiation model for delay described in the previous section.

Since traffic marked as real-time gets prioritized treatment compared to elastic traffic, the cost to use the high priority queue, which is represented by the fair label L^{rt}_{fair} above which we start having losses, should always be higher for real-time traffic than for elastic traffic. However, L^{rt}_{fair} and L^{el}_{fair} are computed independently as a function of the offered load of real-time and elastic traffic, respectively. Therefore, in the case when there is a much higher density of elastic traffic than real-time traffic in the network, we have a situation where $L^{el}_{fair} < L^{rt}_{fair}$ — i.e. elastic traffic would be more expensive even though it receives a poorer service[1]. In order to correct this undesirable situation, we define a new fair label for real-time traffic,

$$\widetilde{L^{rt}_{fair}} = min\left(L^{rt}_{fair}, L^{el}_{fair}/p\right) \tag{5.2}$$

where p is a constant greater than 1, which is used to express the difference in price between a real-time packet and a packet marked as elastic traffic: thus, for the same price, a user will be able to send p times more traffic marked as elastic than traffic marked as real-time[2]. This is the price that the user has to pay in order to guarantee a low delay.

5.3 User-based Admission Control

So far we have concentrated on the impact of delay into the performance of real-time applications. In the following we focus on the bandwidth requirements.

Traditionally, video and audio applications have been designed with hard real-time requirements. Such applications have an intrinsic bandwidth requirement and the performance degrades badly as soon as the bandwidth share becomes smaller than the intrinsic generation rate. This is illustrated in the utility curve of Figure 5.3.

There is another type of real-time applications. Rate-adaptive applications adjust their transmission rate to the available bandwidth. The performance of these applications increases with bandwidth. However, there is a minimum bandwidth below which the signal quality is unbearably low and, as a consequence, utility is practically null. Also, there is a maximum bandwidth above which the marginal utility of additional bandwidth is very slight because the signal quality is much better than humans need. This is illustrated in the utility curve of Figure 5.4.

[1]Note that in this situation real-time traffic would not starve elastic traffic, since real-time traffic is restricted to use a maximum capacity of $\frac{1}{k} \cdot C$, but it would be eating up bandwidth at a lower cost than elastic traffic.

[2]The idea of giving a better treatment in terms of delay but worse in terms of bandwidth to real-time traffic has also been used in the Asymmetric Best-Effort Service proposal [57].

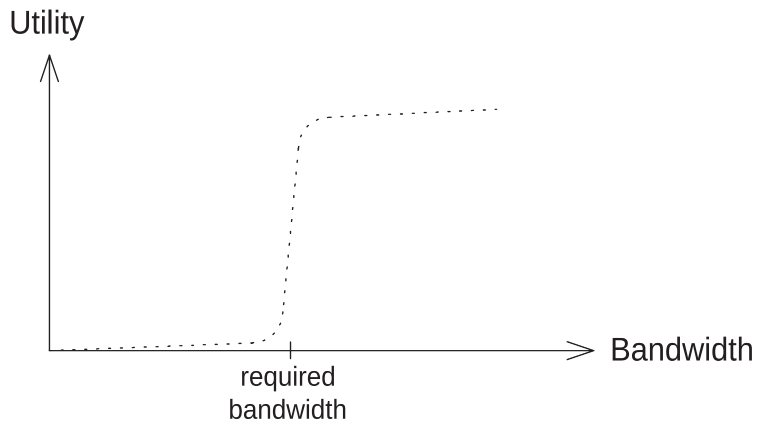

Figure 5.3: Utility function of hard real-time applications as a function of the bandwidth.

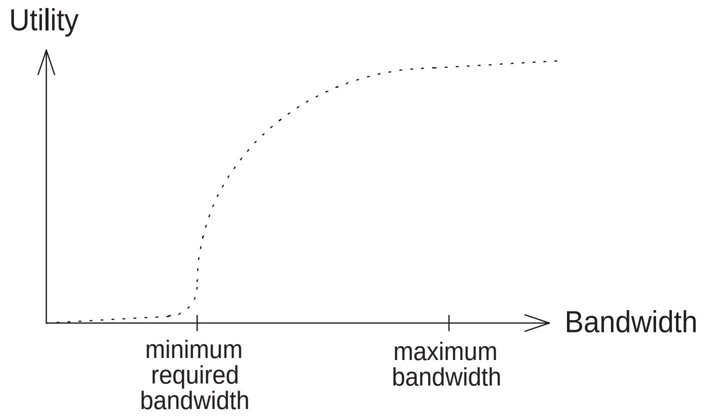

Figure 5.4: Utility function of rate-adaptive real-time applications as a function of the bandwidth.

The problem of transmitting real-time streams over the Internet has traditionally been dealt with two different approaches.

One approach is to adapt application behavior to the best effort service provided by the network, i.e. to the time-varying characteristics of the channel over which the application data packets are sent. We call this approach *adaptive applications*. This adaptation is done by incorporating to the application control mechanisms that attempt to minimize the negative impact of channel characteristics on the quality of the data delivered at the destination. Examples of such mechanisms include playout adjustment (to control delay jitter), rate control (to match bandwidth requirements to available bandwidth), error recovery (to control the impact of packet loss on quality), etc. The drawback of this approach is that, since it is not possible to enforce that bandwidth is above the minimum required, we risk a null performance (see Figure 5.4).

The other approach is to augment the best effort service with other services providing various degrees of performance guarantees. This is done by accepting only those requests that can be served with the desired quality, while rejecting the others. We call this approach *controlled access*. This approach is the one taken by IntServ and DiffServ. The drawback of these architectures is that, in order to provide performance guarantees (e.g. a given bandwidth), they need some kind of network-based admission control, which leads to a high level of complexity, as we have already mentioned before in this thesis.

In this section we propose an approach to the problem of bandwidth allocation for real-time traffic in RT-UFD that combines the two solutions described above. We have called the proposed approach *User-based Admission Control* (UBAC). The novelty of UBAC is that it is user-based instead of network-based. The fact that the network is unaware of the admission control makes things much simpler, overcoming thus the complexity of network-based admission control schemes such as DiffServ or IntServ.

The proposed mechanism relies on measurement-based admission control. L_{fair}, which is used at each link to determine the throughput that each user receives, provides a natural basis for measurement-based admission control: the throughput of a user's flow is determined by the most restrictive (i.e. lowest) L_{fair} on the flow's path: the minimum fair label.

In order to convey the information of the minimum fair label to the sender we use a similar procedure to [42]. We let another piece of information, in addition to the packet's label L_k, travel with the packets: L_{fair}^{min}. L_{fair}^{min} is initialized at the sender with a value equal to ∞[3] and is updated on each

[3]An ∞ value of L_{fair}^{min} is actually represented by 0. Note that a L_{fair}^{min} value of 0 is never

congested link according to the following equation:

$$L^{min}_{fair_{new}} = min(L^{min}_{fair_{old}}, L_{fair_{link}}) \tag{5.3}$$

where $L^{min}_{fair_{old}}$ is the value of L^{min}_{fair} carried by the packet before reaching the link, $L^{min}_{fair_{new}}$ is the value of L^{min}_{fair} after having passed the link and $L_{fair_{link}}$ is the link's fair label.

With the above, the minimum fair label information, contained in the L^{min}_{fair} field of each received packet, reaches the receiver. This information is then piggybacked to the sender using the next control packet sent to him[4].

The sending user can then use that minimum fair label information for each flow to adaptively assign to the flows the necessary weight W_i to reach their destinations without drops:

$$W_i \geq \frac{r_i^{send}}{s_u \cdot L^{min}_{fair}} \tag{5.4}$$

Note that the weights W_i of a user suffer the restriction

$$\sum_{i \in U} W_i \leq 1 \tag{5.5}$$

Using Equation 5.5, a user can perform admission control in such a way that if a rate r_i^{send} is requested for a path that requires a weight W_i that violates Equation 5.5, then the request is rejected[5].

The main advantage of the UBAC scheme is its simplicity. This simplicity results from the fact that it is user-based and therefore does not require complex network mechanisms. However, a restriction is that a user-based admission control scheme like the proposed cannot control the absolute congestion level of the network. As a consequence, it is not possible to guarantee that the bandwidth committed to a flow will be available during the whole flow's lifetime. Note that this restriction is inherent to all measurement-based admission control schemes.

With UBAC, a new request is accepted only when the current level of congestion is low enough. This relates to the *controlled access* approach described above. However, the congestion level may increase after the request

used otherwise.

[4]UBAC requires some exchange of control packets between the sender and the receiver before the sender can decide whether to accept or not a request towards this receiver. The protocol used between the sender and the receiver for requesting resources and notifying the acceptance/rejection of the requests could be based, for example, on RSVP [58].

[5]Note that a safety margin can be used in the admission control decision. The impact of the safety margin on the resulting quality has been studied via simulation in Section 5.4.3.

has been accepted. In order to adapt to the varying bandwidth available for him, a user needs to be able to modify the sending rate of his applications. This relates to the *adaptive applications* approach.

The fact that with the proposed user-based admission control scheme a new request is only accepted when the current level of congestion is low enough guarantees that the probability that the bandwidth available for an application decreases below its minimum bandwidth required is very low. This guarantee represents a fundamental advantage with respect to the *adaptive applications* approach. In Section 5.4.3 we study this issue via simulation.

In the literature, a number of distributed measurement-based admission control schemes have been proposed [59, 60, 61, 62]. These schemes, like UBAC, also accept or reject a new request based on the measured level of congestion. A fundamental difference between these schemes and ours is that these schemes, in contrast to UBAC, require some degree of participation of the network. For example, with these schemes signaling is usually performed by the network. Also, billing is necessarily performed on a usage or reservation basis, since otherwise a user could have permanent reserved paths at no cost by constantly sending useless traffic through them. Note that, in contrast, with our approach, billing is performed on a flat rate basis, based on the share contracted. To the knowledge of the author, the UBAC scheme is unique in that it can provide a service commitment to flows without the network participation.

It is also important to note that the solution presented in this section introduces a new reservation paradigm. With traditional reservation schemes, a call request from a user in a channel with capacity for 5 calls would be rejected if these 5 calls are currently occupied by a second user. This raises the fairness issue of whether it is fair that a user's request is rejected because of another user's resource-hungry behavior. In addition, this leads to an unfair resource distribution, since one user is not allowed to consume network resources because of another user who is consuming a lot of them. In contrast to traditional reservation schemes, with UBAC, the above situation would be solved in such a way that the call of the first user is accepted and served with full quality, at the expense of decreasing the quality of the 5 calls of the second user.

Finally, note that the use of the UBAC solution proposed in this section is optional for the user. Using UBAC, a user can concentrate his contracted share among a few prioritized flows upon congestion, in order to provide them with the required bandwidth, and thus avoid distributing the share among too many flows. However, it is up to the user how to use the resources that he has been contracted through his share. The user can choose to use some policy for user-based admission control to provide service commitment to

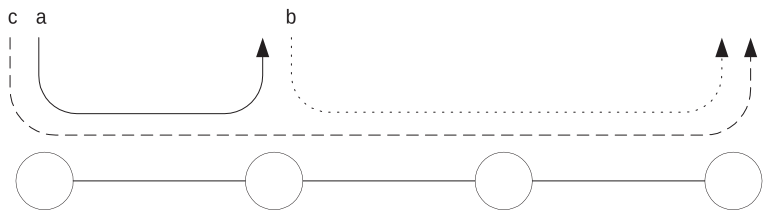

Figure 5.5: Network topology used for the simulations.

some of his flows while rejecting service to others, or can just choose to use the resources in a best-effort way.

5.4 Simulations

In this section we evaluate via simulation the performance of the RT-UFD extension presented in this chapter.

5.4.1 Bandwidth and Delay Distribution

To evaluate the resulting bandwidth and delay distributions with RT-UFD, we simulated the proposed extension on a network as shown in Figure 5.5. This network has three 10 Mbps inter-node links, with a propagation delay of 1ms each. There are twelve users, each sending one flow. The flows travel along different network paths (paths a, b and c illustrated in Figure 5.5). We simulated three different scenarios: one with real-time users sending CBR traffic, another with bursty traffic and a third with mixed CBR and bursty traffic. All elastic users sent background CBR traffic. The traffic characteristics for each simulation are given in Table 5.1. Traffic of bursty users consisted of an aggregation of 10 ON/OFF sources. The active periods of the ON/OFF sources were Pareto distributed with 50 ms average, while idle periods were exponentially distributed with 50 ms average. The values of k and p used were $k = 2$ and $p = 10$, meaning that real-time traffic was twice as expensive as elastic traffic and that the bandwidth used by real-time traffic in a link was limited to one tenth of its capacity.

Table 5.2 shows the throughput (r), average delay (m_d) and standard deviation of the delay (σ_d) experienced by each of the users. Simulation results demonstrate that the expected behavior is reasonably achieved.

For real-time traffic, delays are close to propagation and transmission delays; queuing delays and, consequently, jitters, are negligible[6]. In contrast,

[6]Note that queuing delay is the only variable component of delay and thus the only

				scenario 1		scenario 2		scenario 3	
user	type	share	path	source	avg. rate (Mbps)	source	avg. rate (Mbps)	source	avg. rate (Mbps)
1	real-time	1	a	CBR	0.5	bursty	0.5	bursty	0.5
2	real-time	1	b	CBR	0.5	bursty	0.5	CBR	0.5
3	real-time	1	c	CBR	0.5	bursty	0.5	bursty	0.5
4	real-time	2	a	CBR	0.5	bursty	0.5	CBR	0.5
5	real-time	2	b	CBR	0.5	bursty	0.5	bursty	0.5
6	real-time	2	c	CBR	0.5	bursty	0.5	CBR	0.5
7	elastic	1	a	CBR	5	CBR	5	CBR	5
8	elastic	1	b	CBR	5	CBR	5	CBR	5
9	elastic	1	c	CBR	5	CBR	5	CBR	5
10	elastic	2	a	CBR	5	CBR	5	CBR	5
11	elastic	2	b	CBR	5	CBR	5	CBR	5
12	elastic	2	c	CBR	5	CBR	5	CBR	5

Table 5.1: Traffic Characteristics.

		scenario 1			scenario 2			scenario 3		
user	$r_{theoretical}$ (Kbps)	r (Kbps)	m_d (ms)	σ_d (ms)	r (Kbps)	m_d (ms)	σ_d (ms)	r (Kbps)	m_d (ms)	σ_d (ms)
1	167	167	2.08	0.34	172	2.21	0.39	247	2.41	0.66
2	167	198	3.99	0.31	198	3.91	0.27	204	4.02	0.47
3	167	163	6.28	0.63	155	6.28	0.55	198	6.49	0.54
4	333	339	2.62	0.64	340	2.48	0.58	414	2.77	0.63
5	333	374	4.30	0.50	359	4.06	0.41	381	3.98	0.42
6	333	321	7.09	0.84	320	6.68	0.71	354	6.76	0.75
8	1500	1540	22.30	11.78	1541	33.83	11.67	1577	52.39	6.93
7	1500	1507	36.89	8.92	1516	32.14	10.39	1498	53.45	6.27
9	1500	1440	59.97	13.48	1459	66.59	16.88	1425	106.35	9.76
10	3000	3019	22.65	11.77	3040	33.98	11.72	3131	52.64	6.90
11	3000	3008	37.43	8.86	2972	32.55	10.55	2943	53.84	6.27
12	3000	2918	60.61	13.59	2923	67.32	16.61	2730	106.67	9.78

Table 5.2: Bandwidth and Delay Distribution.

jitter for elastic traffic is much higher (approximately, one order of magnitude higher). We conclude that the UFD-RT extension is effective in implementing the Step Differentiation model for delay.

The throughput experienced by each user is approximately proportional to his share, as stated by the Proportional Differentiation model for bandwidth. Simulation results, however, show a favorable treatment for those flows traversing shorter (a) and less congested (b) paths. Bandwidth between real-time and elastic traffic is shared according to the k value chosen: 9 Mbps for elastic traffic and 1 Mbps for real-time traffic in each link.

one that has effect on the jitter.

		scenario 1			scenario 2			scenario 3		
user	$r_{theoretical}$ (Kbps)	r (Kbps)	m_d (ms)	σ_d (ms)	r (Kbps)	m_d (ms)	σ_d (ms)	r (Kbps)	m_d (ms)	σ_d (ms)
1	79	75	2.19	0.39	78	2.01	0.29	77	2.12	0.41
2	79	89	4.19	0.37	77	3.87	0.26	91	3.97	0.24
3	79	76	6.44	0.56	74	6.22	0.49	75	6.32	0.63
4	159	160	2.43	0.56	156	2.13	0.41	154	2.38	0.57
5	159	160	4.33	0.47	169	3.95	0.34	170	4.05	0.35
6	159	156	6.86	0.70	154	6.40	0.59	148	6.30	0.69
7	1587	1570	26.64	10.22	1573	31.45	8.86	1600	41.36	7.98
8	1587	1611	25.02	10.97	1641	32.82	11.19	1633	32.30	10.42
9	1587	1557	52.30	14.49	1508	64.91	14.27	1564	74.35	14.62
10	3175	3225	25.41	10.93	3241	33.01	11.23	3253	32.51	10.35
11	3175	3173	27.34	10.23	3207	31.88	8.92	3201	41.91	7.96
12	3175	3108	53.28	14.53	3132	65.74	14.27	3067	74.87	14.37

Table 5.3: Pricing for Elastic and Real-Time Traffic.

5.4.2 Pricing for Elastic and Real-Time Traffic

In a second experiment, we used the same scenarios as in the previous experiment to simulate how well it is guaranteed in RT-UFD that real-time packets are priced p times higher than elastic packets. For this purpose, we increased the shares of the users sending elastic traffic from 1 and 2 to 10 and 20, respectively (i.e. we increased the money paid by the elastic users). The other parameters used in the second experiment are identical to the corresponding ones in the first experiment.

Table 5.3 shows the resulting values from the second experiment. It can be seen that both delay and delay deviation expose a very similar behavior as in the first experiment. The throughput distribution however differs. The reason for this is that real-time traffic cannot keep consuming $1/k$ of the link's capacity while still being priced at least p times more than elastic traffic. As a consequence, the bandwidth assigned to real-time traffic is decreased in order to ensure that the ratio s/r (share divided by the rate, which represents the price paid for a unit of bandwidth) is p times higher for real-time traffic than for elastic (0.012 for real-time traffic and 0.006 for elastic). We conclude that the proposed mechanism is effective in pricing real-time traffic higher than elastic traffic.

5.4.3 UBAC

In a third experiment, we evaluated the performance of the User-based Admission Control (UBAC) scheme proposed in Section 5.3. For this purpose we used scenario 3 of Table 5.1 (mixed CBR and bursty traffic scenario) but with users 3 (share of 1) and 6 (share of 2) sending voice traffic and applying

UBAC as described in the following.

Both users 3 and 6 have 10 voice traffic sources. The duration of each voice call is exponentially distributed with average $1/\nu = 120$ s, and the time elapsed between a call end or a failed attempt and the next call attempt is also exponentially distributed with average $1/\lambda$ s. Voice sources adapt to the available bandwidth in the following way. When there is enough bandwidth available, they send full voice quality at a rate of 64 Kbps. Upon congestion, a user reduces the rate (and consequently quality) of some of his calls to 32 Kbps (we consider in this simulation that this is the minimum bandwidth required for an acceptable audio quality). A call that sees its sending rate reduced to 32 Kbps keeps sending at this rate for the rest of its lifetime. If the bandwidth available for a user is such that it is not enough to reduce the sending rate of all his calls to 32 Kbps, some calls are cancelled. Note that this is obviously an undesirable situation that should be avoided with UBAC.

User-based Admission Control (UBAC) is performed by each user according to Algorithm 2 (based on Equation 5.5), where N is the number of active calls from the user and m the safety margin. Unless otherwise specified, we take $m = 0$.

Algorithm 2 UBAC Algorithm.

```
For every new request:
$a = \sum W_i = (N+1)\frac{64Kbps}{L_{fair}^{min} \cdot s_u}$
**if** $a(1+m) \leq 1$ **then**
  accept request
**else**
  reject request
**end if**
```

L_{fair}^{min} is the last minimum fair label received via the control packets. The receiver sends a control packet with the last update of the minimum fair label for every 10 data packets received from the sender.

We define the *offered load* (L) as the average rate at which a user would be sending if all his call attempts were accepted. That is,

$$L = 10 \cdot \frac{\lambda}{\nu} \cdot 64Kbps \tag{5.6}$$

All simulations described in this section have a duration of 1800 s, 100 s of which correspond to warm-up time.

Figure 5.6 shows the throughput experienced by users 6 (share 2) and 3 (share 1) as a function of the offered load. We can observe that at low loads

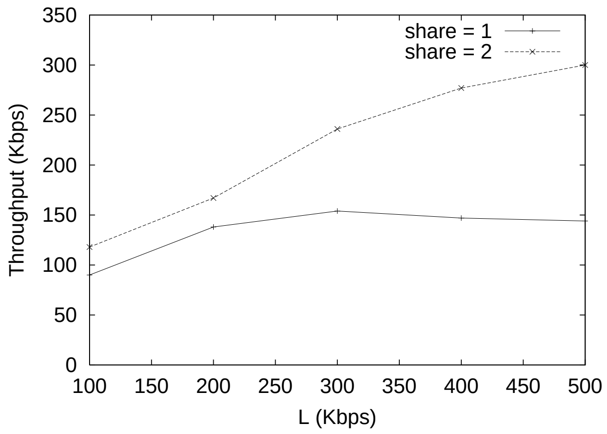

Figure 5.6: UBAC. Throughput.

($L = 100$ Kbps) all calls are accepted and the experienced throughput equals approximately the offered load. At higher loads, some calls are rejected and, as a consequence, the throughput is lower than the offered load. However, the throughput tends asymptotically to user's fair share of bandwidth, which is 167 Kbps for user 3 and 333 Kbps for user 6. We conclude that UBAC allows users to make a reasonably efficient use of their share of bandwidth, while still providing a good service quality to the accepted calls as shown in the following.

Figure 5.7 shows the packet loss probability as a function of the offered load. Since the goal of UBAC is to only accept those calls that can be served with no packet loss, this probability should be kept low. We can observe from the results obtained that this probability is always kept very small (below 1%).

The share contracted by a user should reflect the quality experienced by this user. With elastic traffic, service quality is expressed by the bandwidth, as it has been explained in other chapters. With real-time traffic, assuming that all accepted requests are served with a reasonable quality, the service quality is expressed by the request acceptance rate. Figure 5.8 shows the probability that a request is rejected for the two users (user 3 with a share of 1 and user 6 with a share of 2). Since this probability is much higher with the lower share, we conclude that RT-UFD is effective in delivering a better service quality to those users who pay more (i.e. contract a higher share).

Another parameter that expresses the delivered service quality is the ratio of calls that are served with full quality (64 Kbps) and calls whose quality is degraded during their lifetime (i.e. that are forced to reduce their sending rate to 32 Kbps). Figure 5.9 shows the probability that a call is forced to degrade

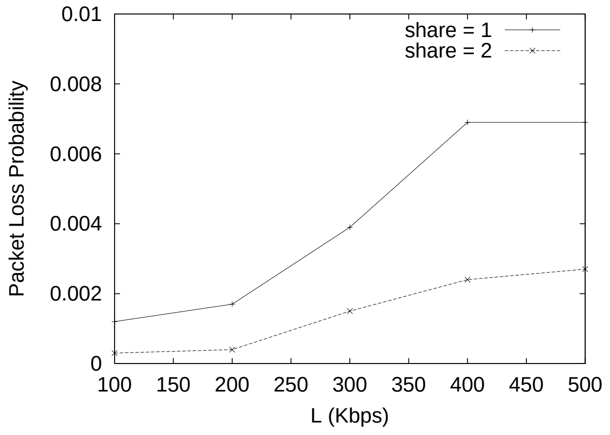

Figure 5.7: UBAC. Packet Loss Probability.

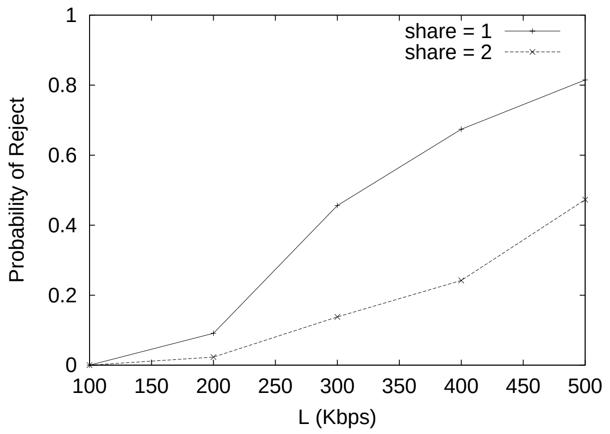

Figure 5.8: UBAC. Probability of Reject.

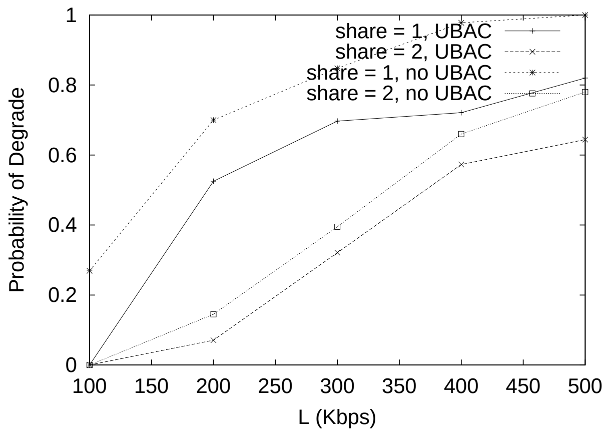

Figure 5.9: UBAC. Probability of Degrade.

its quality as a function of the offered load. To better understand the level of effectiveness of UBAC, we compared these results with the results that we would obtain with the same adaptive sources but no UBAC (i.e. accepting all calls).

It can be observed from the results of Figure 5.9 that the probability that a call is forced to degrade its quality is lower with a higher share, i.e. also in this case a higher share implies a better service quality. Also, this probability is much higher when no UBAC is used, which shows the usefulness of UBAC.

As we have explained before, if the bandwidth available for a real-time application is lower than the minimum required (in our case 32 Kbps), the quality is unacceptable and the call must be cancelled. This is obviously highly undesirable. With UBAC, we never observed a cancelled call in all the simulations performed. In contrast, the probability that a call is cancelled without UBAC can be considerably large at high loads, as can be observed in Figure 5.10.

So far all simulations have been run taking the UBAC parameter safety margin m equal to 0. As it can be seen from Algorithm 2, m expresses the excess available bandwidth margin: a call is only accepted if after accepting it there is still a portion $\frac{m}{1+m}$ of the available bandwidth for this user unused. The objective of keeping this excess available bandwidth is to be able to absorve new situations of congestion without harming the quality of the ongoing calls.

The parameter m, thus, represents a tradeoff between the probability of having a request rejected and the probability of having the quality of an ongoing call degraded. This tradeoff has been studied via simulation by increasing m from 0 to 1 in the previous scenario when the offered load

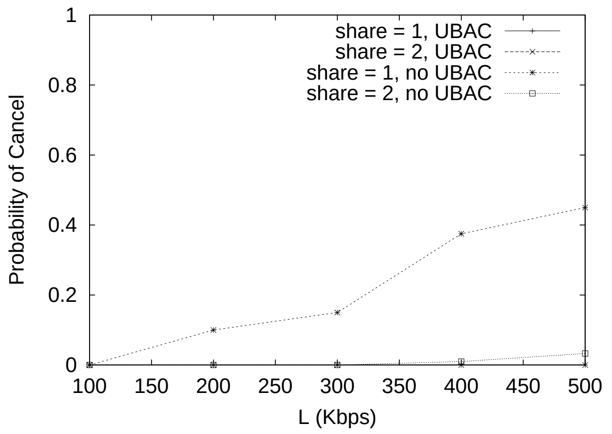

Figure 5.10: UBAC. Probability of Cancel.

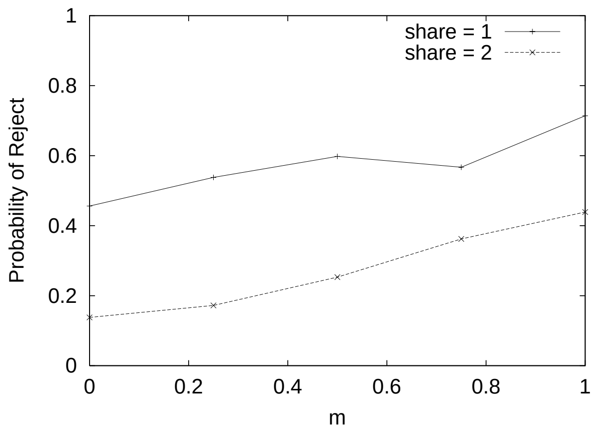

Figure 5.11: Safety margin. Probability of Reject.

is of 300 Kbps. The results obtained are shown in Figures 5.11 and 5.12. We can observe that, as we expected, when increasing m the probability of reject increases, while the probability of degrade decreases. Note that the value of m, as all other UBAC parameters and policies, is chosen by the user depending on his preferences.

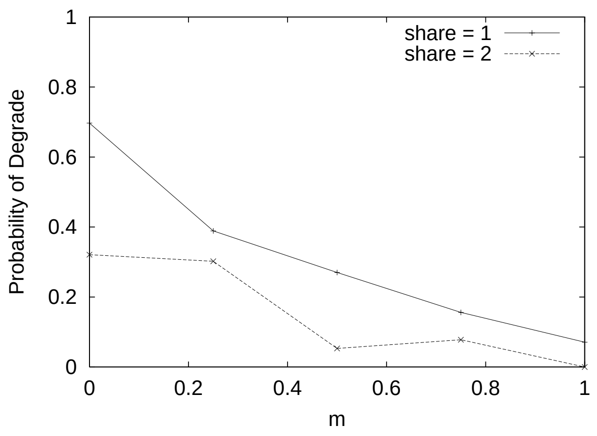

Figure 5.12: Safety margin. Probability of Degrade.

5.5 Summary

UFD-RT extends the UFD architecture with real-time traffic support. This is done in two steps: in a first step we solve the delay issue while in a second step we deal with the bandwidth requirements.

Real-time applications require their packets to arrive earlier than the *play-back* point in order to be played. To adapt to this requirement we have proposed the Step Differentiation criterion for delay. This criterion has proved to be effective in meeting the delay requirements of real-time traffic.

Since real-time packets receive a higher priority treatment than elastic traffic with respect to delay, they should have a higher cost. The traffic type separation mechanism we propose implements this price differentiation by allocating (p times) more bandwidth to a user sending elastic traffic than to a user sending real-time traffic, when both users pay the same price for network resources. The parameter p depends on the pricing policy and is chosen by the network operator.

The bandwidth requirements of real-time applications are met with the User-based Admission Control (UBAC) scheme. This scheme combines two traditional approaches for supporting real-time applications in packet networks: the *adaptive applications* and the *controlled access*. The novelty of UBAC as compared to existing distributed admission control schemes is that no participation from the network is required and signaling between the user and the network is avoided. These features strongly contribute to the simplicity of UBAC.

Chapter 6

Extension for Multicast Traffic

Using multicast delivery to multiple receivers reduces the aggregate bandwidth required from the network compared to using unicast delivery to each receiver. Multicast routing establishes a distribution tree that connects the source with the receivers. The multicast tree is rooted at the sender and the leaves are the receivers. Multicast delivery sends data across this tree towards the receivers. As opposed to unicast, data is not copied at the source, but is copied inside the network at branch points of the tree. The fact that only a single copy of data is sent over a link that leads to multiple receivers results in a bandwidth gain of multicast over unicast whenever a sender needs to send simultaneously to multiple receivers.

During the last decade, multicast routing and delivery have evolved from a pure research topic [63] to being experimentally deployed in the MBONE [64] to being supported by major router manufacturers. Despite the widespread deployment of multicast capable networks, multicast is rarely provided as a service. Among the reasons that slow down the use of multicast we find the lack of a valid service model that defines how to fairly allocate network resources among multicast and unicast flows.

In this chapter we take up previous work on bandwidth allocation for unicast and multicast flows: the Logarithmic Receiver Dependent (*logRD*) policy [65]. This policy gives more bandwidth to a multicast flow as compared to a unicast flow that shares the same bottleneck link, however without starving the unicast flows. We propose an extension of the UFD architecture, the *Multicast UFD* (M-UFD) extension, that implements the *logRD* policy.

6.1 Bandwidth Allocation Policy

For the case when all flows are unicast, the UFD architecture distributes bandwidth in link l among two competing flows i and j of users u and v according to the following equation:

$$\frac{r_i^l}{s_u \cdot W_i} = \frac{r_j^l}{s_v \cdot W_j} \tag{6.1}$$

where r_i^l denotes the bandwidth allocated for flow i in link l.

The above criterion, however, is not appropriate when some of the flows are multicast: it does not seem fair to give the same amount of bandwidth to a flow serving one receiver than to another serving one hundred receivers. We conclude that, in case multicast traffic is present, the UFD architecture should be extended taking into account the number of receivers of each flow.

When choosing a strategy for bandwidth allocation of multicast and unicast flows, our goals are:

Multicast Incentive Since multicast results in bandwidth savings, users should be given an incentive to use it. In general, users want high satisfaction, but do not care whether unicast or multicast is used to deliver the content. If we give more bandwidth to multicast, a multicast user will experience a higher satisfaction than a unicast user, which results in an incentive to use multicast.

Fairness Multicast should not be encouraged at the detriment of the level of fairness among users. In other words, multicast users should not be rewarded in such a way that unicast users see their share of bandwidth drastically reduced, since this would lead to an unfair bandwidth allocation.

In [65] the Logarithmic Receiver Dependent (*logRD*) policy is proposed for bandwidth allocation of unicast and multicast flows. With *logRD*, the share of bandwidth allocated to flow i at link l depends logarithmically on the number of receivers R_i^l of flow i that are downstream of link l[1]. Applying this policy to the bandwidth allocation of Equation 6.1 results in the following allocation[2]:

$$\frac{r_i^l}{s_u \cdot W_i \cdot (1 + ln(R_i^l))} = \frac{r_j^l}{s_v \cdot W_j \cdot (1 + ln(R_j^l))} \tag{6.2}$$

[1]We say that a receiver r of flow i is downstream of link l if the data sent by the source of flow i to receiver r crosses link l.

[2]Equation 6.2 is valid for multicast and unicast flows. In the case of unicast flows, R_i^l is taken as 1, since, by definition, unicast flows only have one receiver.

Since with Equation 6.2 multicast receivers are rewarded with more bandwidth than unicast receivers, the *logRD* strategy gives an incentive to use multicast. In addition, [65] shows via analytical studies and simulations that the *logRD* policy achieves a good tradeoff between multicast incentive and fairness.

6.2 Multicast UFD

In order to implement the bandwidth allocation of Equation 6.2, the M-UFD extension introduces ingress and core relabeling for multicast packets as described in the following.

As we can see in Equation 6.2, the bandwidth allocated to a multicast flow depends on the number of downstream receivers that this flow is serving. At the ingress, after the label control, we introduce the information of the number of receivers of a flow i, R_i, to the label of its packets according to the following relabeling formula:

$$L_k^{new} = \frac{L_k^{old}}{1 + ln(R_i)} \tag{6.3}$$

As a packet is forwarded towards the leaves of the multicast tree, the number of downstream receivers served by the packet changes, and we update the packet label accordingly. Let l be the parent link of a multicast flow at node n, where the multicast packets are copied to c child links $l1, \ldots, lc$ (see Figure 6.1). Let R_i^l be the number of downstream receivers of flow i at link l, and R_i^{lj} the number of downstream receivers at link lj. Then, the label of a multicast packet that is forwarded by node n from a parent link l to a child link lj is updated according to the following formula:

$$L_k^{lj} = \frac{1 + ln(R_i^l)}{1 + ln(R_i^{lj})} \cdot L_k^l \tag{6.4}$$

The M-UFD algorithm, resulting from adding the above ingress and core relabeling functionality to UFD, is illustrated in Figure 6.2. With this algorithm, bandwidth in a congested link l is distributed among two flows i and j competing for bandwidth in this link according to:

$$\frac{r_i^l}{s_u \cdot W_i \cdot (1 + ln(R_i^l))} = \frac{r_j^l}{s_v \cdot W_j \cdot (1 + ln(R_j^l))} = L_{fair}^l \tag{6.5}$$

where L_{fair}^l is the fair label of link l. Note that the above allocation corresponds to the bandwidth distribution stated by the *logRD* policy (Equation 6.2).

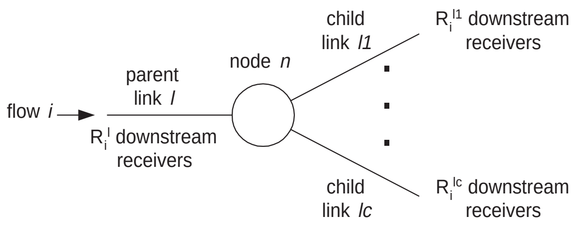
flow i
parent link l
R_i^l downstream receivers
node n
child link $l1$
R_i^{l1} downstream receivers
child link lc
R_i^{lc} downstream receivers

Figure 6.1: Core Relabeling in M-UFD.

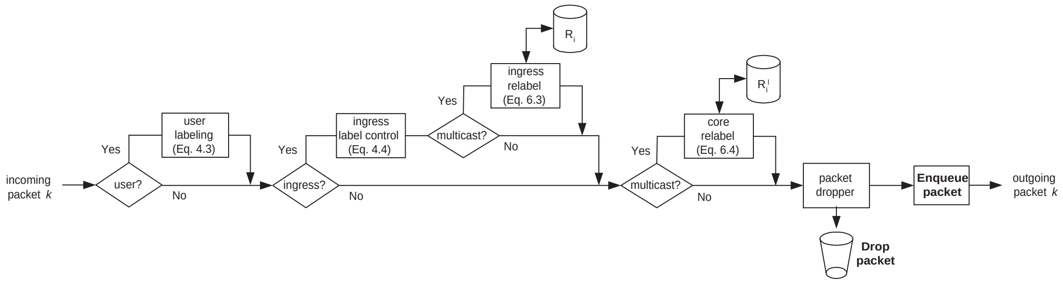

Figure 6.2: M-UFD Algorithm.

In the algorithm of Figure 6.2 it is important to note that M-UFD requires to keep per-flow state for the multicast flows: the estimation of the number of downstream receivers[3] and the multicast routing information. The first is inherent to the *logRD* policy and the second to multicast. In contrast, no per-flow state is required for unicast flows (this was a design goal of UFD). We argue that, since we expect the number of multicast flows crossing a node to be much smaller than the number of unicast flows, keeping per-multicast-flow state does not harm the efficiency of the resulting architecture.

6.3 Layered Multicast

The fact that in M-UFD the bandwidth experienced by a receiver depends on the level of congestion of the links in the path from the sender to the receiver results in an heterogeneity on the amount of bandwidth experienced by the different receivers of the same multicast flow. To cope with this heterogeneity, which is a general problem in multicast transmissions, layered multicast schemes have been proposed.

Layered coders produce a set of streams or layers such that layer 0 provides a minimum quality stream and each layer $i + 1$ adds more quality to

[3]The estimation of the number of downstream receivers is feasible, for instance, with the Express multicast routing protocol [66].

layer i. This feature allows to gracefully adapt the multicast flow's quality to the bandwidth available for each individual receiver, by delivering a different subset of layers to each one.

In this section we extend the M-UFD algorithm to support layered multicast transmissions. In order to have the higher layers dropped first, we introduce a new per-multicast flow variable for each link: the layer threshold y_i^l. The challenge of M-UFD Layered Multicast is to drop layered packets such that:

1. At link l, layers of multicast flow i below the threshold y_i^l are enqueued and layers above are dropped. This leads to having only the lower layers of each multicast flow delivered, which is the objective when using layered schemes.

2. The bandwidth obtained by a multicast layered flow does not exceed the flow's fair share of bandwidth. The flow's fair share of bandwidth is the bandwidth that a flow would receive with the algorithm presented in Section 6.2.

3. Oscillations of the layer threshold y_i^l are minimized. In [10] we have shown that oscillations on the number of layers received harm the resulting video quality.

4. The algorithm to compute y_i^l is efficient and the number of per-multicast flow variables required is minimized.

A traditional approach to ensure that a flow does not exceed its assigned bandwidth is the Token Bucket algorithm. The algorithm we propose for packet dropping of layered multicast is based on this algorithm. Tokens are substracted from the bucket when a packet that would have been dropped according to the algorithm of Section 6.2 is enqued. In the opposite case (i.e. a packet that would have been enqueued with the algorithm of Section 6.2 is dropped) tokens are added to the bucket. Then, by not enqueuing any packet when there are not enough tokens in the bucket for it, we ensure that a flow cannot exceed its fair share of bandwidth (condition 2). As long as there are enough tokens in the bucket, we enqueue layers below y_i^l and drop the rest (condition 1).

The policy we use to compute the value of y_i^l is based on the *static levels policy* that we proposed in [10]. With this policy, the layer threshold is set such that a packet of multicast flow i that belongs to layer y is only enqueued if the bucket occupancy B_i does not exceed a static level B_y (see Figure 6.3). In [10] we have shown that, with a proper configuration of the bucket size

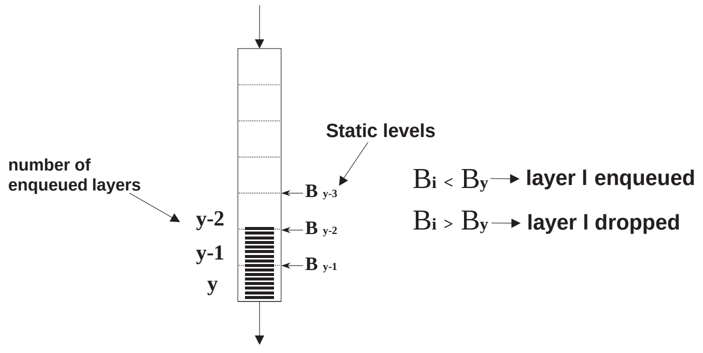
Static levels
number of enqueued layers
y-2
y-1
y
B y-3
B y-2
B y-1
Bi < By → layer l enqueued
Bi > By → layer l dropped

Figure 6.3: Static levels policy.

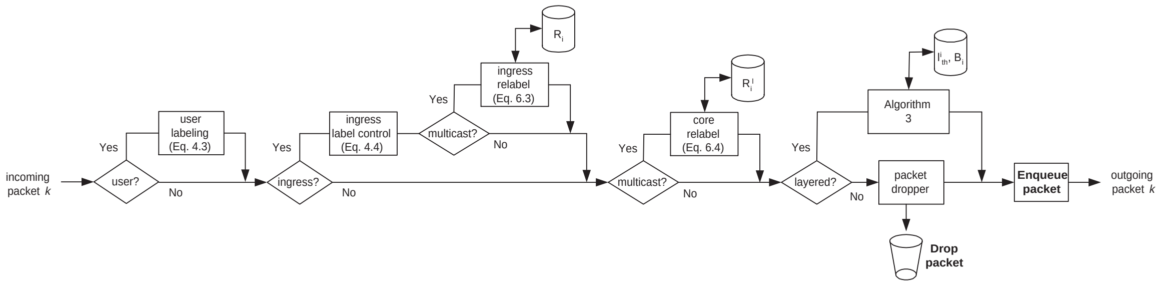

Figure 6.4: M-UFD Algorithm with Layered Multicast.

B_{max}, the *static levels policy* does not result in excessive oscillations on the received number of layers (condition 3).

The resulting algorithm for dropping layered packets is described in Figure 6.4[4] and Algorithm 3. This algorithm only requires to keep one per-multicast variable, the bucket occupancy B_i, in addition to the layer threshold y_i^l (condition 4).

6.4 Experimental Results

To evaluate the performance of the M-UFD architecture for bandwidth allocation and layered multicast, we performed some tests with the implementation explained in Chapter 8.

We used the following configuration that is shown in Figure 6.5. The testbed comprised a PC as core router (AMD-K6 CPU at 350 MHz), two PCs as sender/ingress (Pentium CPU at 200 MHz) and one PC as receiver (AMD-

[4]In the algorithm of Figure 6.4, the information of whether a multicast packet is layered or not and, in the former case, the layer to which the multicast packet belongs, is carried by the RTP header extension that we propose in [10].

Algorithm 3 Dropping Algorithm for Layered Multicast Flows.

```
Upon receiving packet $k$:
$y_k$ = read_layer(packet $k$)
$L_k$ = read_label(packet $k$)
$l_k$ = get_length(packet $k$)
estimate $L_{fair}$
$prob = max(0, 1 - \frac{L_{fair}}{L_k})$
$y_i^l$ = layer_threshold($B_i$)
**if** $prob >$ unif_rand$(0,1)$ **then**
   drop = 1
**else**
   drop = 0
**end if**
**if** $y_k > y_i^l$ **then**
   drop(packet $k$)
   **if** drop = 0 **then**
      $B_i = max(B_i + l_k, B_{max})$
   **end if**
   return
**else**
   **if** drop = 1 **then**
      **if** $B_i < l_k$ **then**
         drop(packet $k$)
         return
      **else**
         $B_i = B_i - l_k$
      **end if**
      enque(packet $k$)
      return
   **end if**
**end if**
```

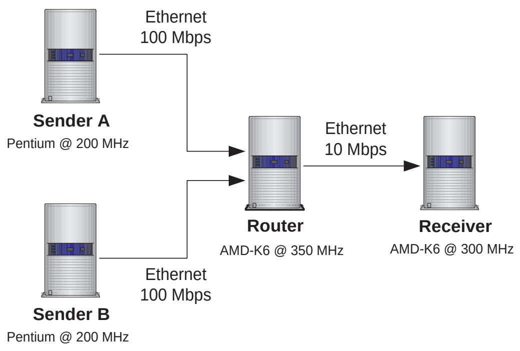

Figure 6.5: Configuration of the test network.

K6 CPU at 300 MHz). The network was build of separate Ethernet segments (two 100 Mbps and one 10 Mbps). The latter segment, that connected the router with the receiver, was the bottleneck link.

6.4.1 Bandwidth Allocation

The bandwidth allocated to a multicast flow with the *logRD* policy increases logarithmically with the number of receivers. In order to evaluate the level of accuracy achieved by the M-UFD architecture in the bandwidth allocation, we performed the following test. Senders A and B consisted of one user each (user A and user B) sending a multicast and a unicast flow respectively at a constant rate equal to 10 Mbps. The number of receivers for the multicast flow (user A) varied from 1 to 100. Both users had a share of 1. The packets length was set to 1000 bytes.

Figure 6.6 shows the results obtained for the above test (throughput obtained by the multicast flow). It can be observed that the results obtained are surprisingly accurate. We conclude that the M-UFD architecture provides the desired bandwidth allocation.

6.4.2 Layered Video

In Section 6.3 we have proposed a dropping algorithm for the transmission of layered flows that adapts the quality of the delivered stream to the bandwidth available for each receiver. The design goal was to find a dropping algorithm that lead to a graceful degradation of the flow's quality while preserving the original bandwidth allocation.

In order to evaluate the performance of the proposed approach we ran some experiments with layered video. We compared the quality of the re-

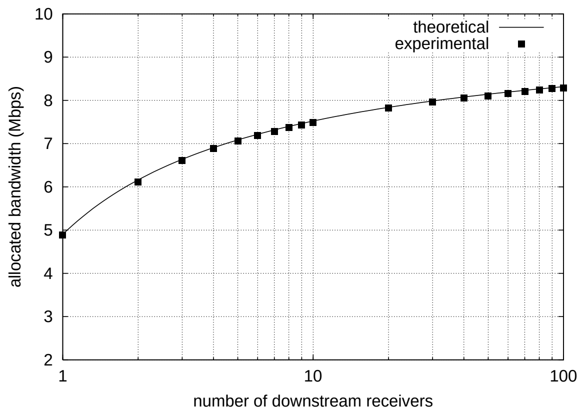

Figure 6.6: Bandwidth Allocation with CBR traffic.

ceived video when using the dropping algorithm explained in Section 6.3, which we refer to as *layered* dropping, and the dropping algorithm of Section 6.2, which we refer to as *uniform* dropping. Experiments were done using the DCT-based layered video codec described in [10]. The original video was coded using 10 layers.

To quantify the level of corruption of the received videos due to packet losses, we used the QMeasure metric described in [67]. This measure tries to estimate the distortion perceived by the human visual system and is known to be more precise than objective measures like the PSNR or the MSE.

In the test we performed, Sender A consisted of one user transmitting a multicast layered video flow, while Sender B consisted of $N-1$ users sending one CBR flow each. The aggregate rate of the CBR flows was of 15 Mbps. All flows had only one receiver and all users had a share of 1. The size of the bucket at the router was set to $B_{max} = 100$ Kbits.

Figure 6.7 shows the resulting bandwidth allocation with uniform and layered dropping. This result validates the dropping algorithm we proposed in Section 6.3, since the bandwidth obtained is the same for both dropping algorithms. This bandwidth is approximately the flow's fair share of bandwidth.

Figure 6.8 shows how the bandwidth delivered for the video flow is distributed among the different layers with both dropping algorithms when $N = 20$. It can be observed that layered dropping algorithm achieves a good level of discrimination among layers, as desired. Note that with uniform dropping, there is no discrimination since the dropping does not distinguish between layers.

Figure 6.9 shows the benefit obtained with layered dropping as com-

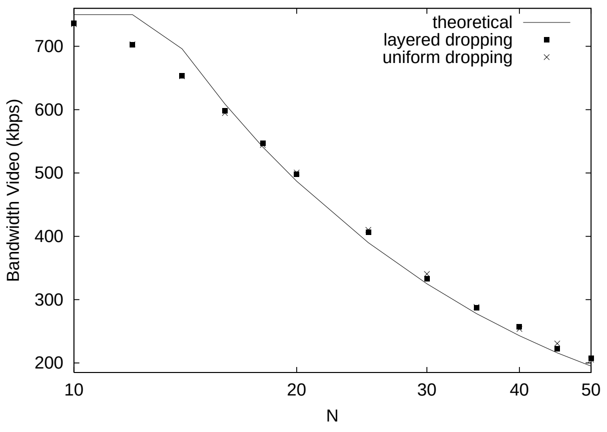

Figure 6.7: Bandwidth Allocation with video traffic.

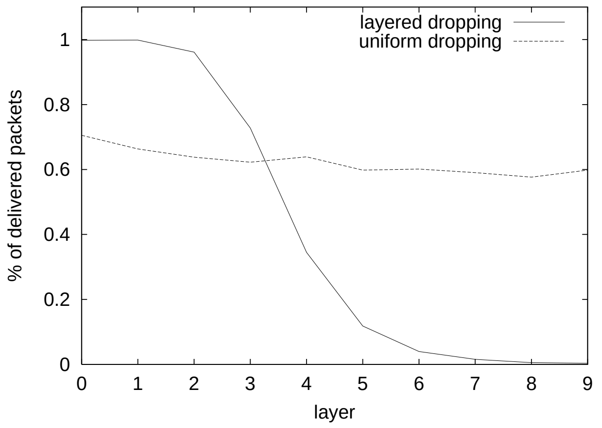

Figure 6.8: Percentage of delivered packets as a function of the layer.

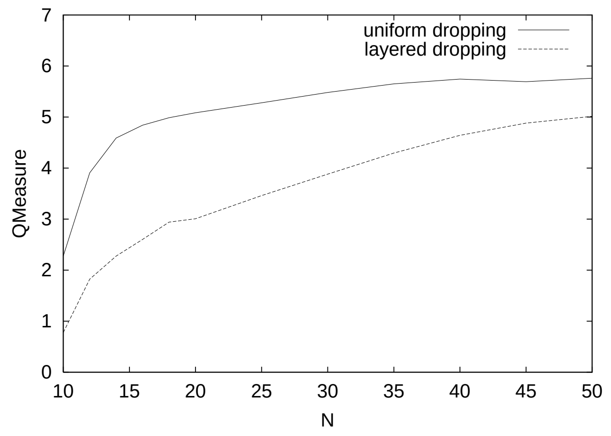

Figure 6.9: Perceived video quality.

pared to uniform dropping. In this figure, video quality is measured with the QMeasure metric explained above. This metric is such that the lower the QMeasure, the higher the video quality. We can observe that, with layered dropping the perceived quality is degraded gracefully (i.e. the QMeasure keeps lower) as the number of competing users (N) increases. In contrast, with uniform dropping the quality suffers a sharp degradation when the bandwidth allocated to the multicast layered flow becomes smaller than the flow's sending rate.

The results reported by the QMeasure match our subjective perception of the videos. With the uniform dropping, when reducing the allocated bandwidth, we observed a jerky display where the movement of objects was corrupted (because of the loss of entire frames), overlapping of images and frequent changes of definition within the same frame. In contrast, with the layered labeling, we only observed a smooth decrease in the definition. This can be observed in the videos that have been made available at `http://www.ccrle.nec.de/projects/mufd/`.

6.5 Discussion and Related Work

One of the reasons why multicast is not currently available in the Internet is because users lack an incentive to use it. The *logRD* policy for bandwidth allocation provides a solution to the problem of encouraging the use of multicast while keeping a good level of fairness between unicast and multicast.

In this chapter we have proposed a multicast extension to the UFD architecture based on the *logRD* policy: the M-UFD extension. M-UFD preserves the feature of UFD of avoiding per-flow or per-user state at core nodes for unicast flows. In contrast, multicast flows require inherently some per-flow state at core nodes. However, since the number of multicast flows is expected to be small as compared to unicast, the resulting architecture still scales well.

M-UFD has been designed to allow a multicast layered flow to accommodate its quality to the available bandwidth of each receiver. The advantage of layered multicast support in M-UFD as compared to other approaches is that it does require neither signaling nor the interaction of the receiver, it reacts immediately upon changing networks conditions and it does not suffer from stability problems.

A related work that extends core stateless fair queuing architectures (specifically CSFQ [33]) to multicast is *mCorelite* [68]. There are a number of fundamental differences between the approach taken by M-UFD and *mCorelite*. The first difference is in the service model. *mCorelite* allocates in a link the same bandwidth for all flows, regardless whether they are unicast or multicast and the number of receivers. We argue that, in order to encourage the use of multicast, users should be given an incentive to use it. The second difference is architectural: *mCorelite* receivers are required to signal the number of layers to be received. We believe that the complexity involved with the signaling is a main a drawback of *mCorelite* as compared to our approach. Finally, *mCorelite* has only been evaluated via simulations and not with a real implementation. The experimental results reported in this chapter, including the test with layered video, are one of the main contributions of the M-UFD extension.

A well known approach for layered multicast is the receiver-driven layered multicast (RLM) [69]. With RLM, the number of layers delivered by the network is updated dynamically with join and leave experiments according to the behavior experienced by the receiver. Some important drawbacks of RLM as compared to our solution are its slow response to network congestion and its instability problems (see [70] for a detailed explanation of these drawbacks).

[71] proposes an alternative bandwidth allocation criterion to the *logRD* policy for multicast and unicast flows. [71], however, does not give users an incentive to use multicast.

Chapter 7

Wireless UFD

Resource Allocation in wireless networks has a special relevance due to the scarce resources available in such networks. Since wireless networks may be considered as just another technology in the communications path, it is desirable that the architecture for resource allocation follows the same principles in the wireless network as in the wireline Internet, assuring compatibility among the wireless and the wireline parts.

In this chapter we address this issue by extending the MAC protocol of the IEEE 802.11 Wireless LAN standard. The challenge in Wireless LAN is that we do not have all packets in a centralized queue, like in wired links, but we have them distributed in the wireless hosts. Therefore, we need a MAC mechanism capable of providing the desired scheduling.

The architecture we propose for Wireless LAN, the *Wireless UFD* architecture, allocates resources in the Wireless LAN following the same principles as the UFD architecture for wired links.

The rest of the chapter is structured as follows. We first introduce the state of the art; we recall the basics of the IEEE 802.11 standard and present a review on related work. Then, we introduce the proposed architecture as an extension of the 802.11 standard. The performance of the architecture is thoroughly evaluated via simulation. The chapter closes with a summary.

7.1 State of the Art

7.1.1 The IEEE 802.11 MAC layer

The basic IEEE 802.11 Medium Access mechanism is called Distributed Coordination Function (DCF) and is based on the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol [72]. CSMA/CA was first investigated thoroughly in [73] and [74]. The MAC scheme used in IEEE 802.11 is an extended version of the FAMA protocol [75]. It is slotted, i.e. the access can happen only at specific instants. The 802.11 MAC protocol operation is shown in Figure 7.1.

In the DCF mode, a station must sense the medium before initiating the transmission of a packet. If the medium is sensed idle for a time interval greater than the DCF Inter Frame Space (DIFS), then the station transmits the packet. Otherwise, the transmission is deferred and a backoff process is started.

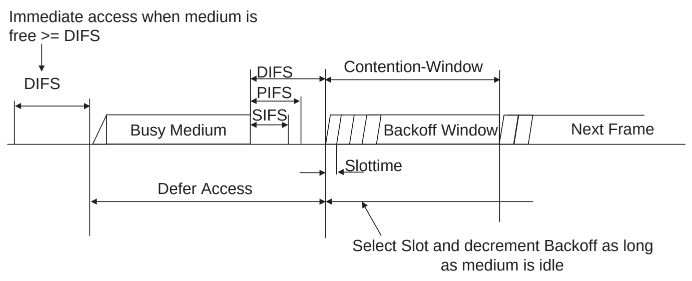
Immediate access when medium is free >= DIFS
DIFS
DIFS
PIFS
SIFS
Contention-Window
Busy Medium
Backoff Window
Next Frame
Slottime
Defer Access
Select Slot and decrement Backoff as long as medium is idle

Figure 7.1: Basic 802.11 MAC protocol operation

Specifically, the station computes the backoff interval as an equally distributed random value taken from the range of 0 to the so-called Contention Window (CW), where the backoff time is measured in slot times. This backoff interval is then used to initialize the backoff timer. This timer is decreased only when the medium is idle and is frozen when it is sensed busy. Each time the medium becomes idle for a period longer than a DIFS, the backoff timer is periodically decremented, once every slot-time.

As soon as the backoff timer expires, the station starts to transmit. A collision occurs when two or more stations start transmission simultaneously in the same slot. To avoid collisions, a Request To Send (RTS) and a Clear To Send (CTS) can be exchanged between source and receiving stations prior to the actual frame transmission. In addition, an Acknowledgement (Ack) is transmitted to the source after successful reception of the frame to detect collisions. The Ack scheme can additionally be used to control the retransmission of erroneous frames. The RTS/CTS scheme is also used for hidden node handling.

If a CTS or acknowledgment is not received by the source station, it assumes that the transmission attempt was not successful and re-enters the backoff process. To reduce the probability of collisions, the CW is doubled after each unsuccessful transmission attempt until a predefined maximum

(CW_{max}) is reached. After a successful frame transmission, if the station still has frames buffered for transmission, it must execute a new backoff process.

The second access mechanism specified in the IEEE standard is built on top of DCF and it is called Point Coordination Function (PCF). It is a centralized mechanism, where one central coordinator polls stations and allows them undisturbed, contention free access to the channel. In contention free access mechanism, collisions do not occur since the access to the channel is controlled by one entity. This scheme has no practical meaning to this thesis, except that the access is prioritized due to the shorter PCF Inter Frame Space (PIFS).

The three Inter Frame Spaces (IFS) serve the purpose of defining different levels of access priorities. They define the minimal time that a station has to let pass after the end of a frame, before it may start transmitting a certain type of frame itself. After a SIFS (Short IFS), the shortest interframe space, only acknowledgements, CTS and data frames in response to poll by the PCF may be sent. The use of the DIFS and the PIFS has already been explained. This use of IFS allows the most important frames to be sent without additional delay and without having to compete for access with lower priority frames.

7.1.2 Related Work

Current trends in wireless networks indicate a desire to provide a flexible wireless infrastructure that can support emerging multimedia services along with traditional data services.

One possible approach for supporting multimedia in Wireless LAN is based on the Integrated Services architecture proposed for the wireline Internet [76]. In this approach, the control over wireless resources is very strict, motivated by the argument that strict control, with complex and sophisticated mechanisms and protocols, is required to maintain good quality in the wireless environment.

Another approach for multimedia support in Wireless LAN is based on the Differentiated Services architecture, which provides service differentiation using more simple mechanisms. There have been several proposals for service differentiation in wireless networks, like in [77]. These mechanisms, however, rely on centralized control and polling of backlogged mobile hosts. In contrast to these proposals, the architecture we propose in this chapter is based on distributed control. We argue that distributed control results in a more productive use of radio resources.

[78], [79], [80], [81] and [82] are other proposals for service differentiation

relying on distributed control. These architectures are based on the idea of modifying the backoff time computation of the 802.11 standard to provide service differentiation, which is also the basis of our elastic traffic extension.

In [78] the backoff time computation is modified by assigning shorter CWs to real-time traffic. The main difference between [78] and our proposal is that [78] does not decouple real-time traffic from elastic traffic and, as a consequence, the service quality of real-time traffic in [78] is sensitive to the changing conditions of elastic traffic. In addition, [78] does not provide different priorities for elastic traffic.

[79] and [80] propose the use of different CWs and different backoff increase parameters, respectively, for different priorities in elastic traffic, and they do not support real-time traffic. The fact that the parameters in [79] and [80] are statically set makes the resulting bandwidth distribution uncertain, as opposed to our proposal, in which the desired bandwidth allocation is achieved by modifying dynamically the CWs considering both the aggressiveness of the sources and their willingness to transmit.

The idea of modifying dynamically the backoff computation parameters had already been mentioned in [81]. However, [81] provides neither an algorithm nor simulation results for this dynamic adaptation.

[82] provides relative priorities for delay and throughput in a multi-hop wireless network. This approach piggybacks scheduling information onto RTS/DATA packets and then uses this information to maintain a scheduling table in each node. This table is then used to modify the computation of the backoff times. One major drawback of [82] as compared to our approach is its complexity. Moreover [82] does not provide backwards compatibility.

The Black Burst scheme in [83] introduces a distributed solution to support real-time sources over 802.11, by modifying the MAC for real-time sources to send short transmissions to gain priority. This method can offer bounded delay. The disadvantage of [83] is that it is optimized for isochronous sources, preferably with equal data rates, which can be a significant limitation for applications with variable data rates.

The basic access mechanism of 802.11, the DCF mode, does not guarantee anything else than Best Effort service. The second access mechanism specified in 802.11, the PCF mode, is intended to support real-time services by using a central polling mechanism. This mechanism, however, is not supported in most wireless cards, and it was shown in [84] that the cooperation between PCF and DCF modes leads to poor throughput performance.

The IEEE 802.11 Working Group has already identified the need for QoS enhancements, and has created a special Working Group dealing with such issues. Most of the proposals presented to this Working Group are based on centralized mechanisms, and redefine in one or another way the Point

Coordination Function. NEC has participated in this standardization effort with its own proposal [18, 19]. The Wireless UFD architecture of this chapter is based on this proposal.

In contrast to 802.11, HIPERLAN/1 [85] does support delivery of packets with different priorities, which is achieved by a scheme similar to the different IFSs used in IEEE 802.11. The MAC protocol used in HIPELRAN/1 is EY-NPMA (Elimination Yield – Non-preemtive Priority Multiple Access). The Contention Resolution Algorithm in our real-time traffic extension (CRA-RT) uses concepts of the EY-NPMA protocol.

The HIPERLAN/2 standard has recently been published [86]. It uses a centrally controlled MAC scheme. An international harmonization of the IEEE 802.11 and HIPERLAN/2 standards is ongoing at present.

7.2 Architecture

The goal of the Wireless UFD architecture is to extend the resource allocation of the UFD architecture to the user's Wireless LAN[1]. In order to provide the desired resource allocation in a Wireless LAN we have to:

- Ensure that real-time packets corresponding to requests that have been accepted by the user's admission control are forwarded in the Wireless LAN with a very low delay.
- Distribute the remaining bandwidth available in the Wireless LAN among elastic flows proportionally to the weights W_i assigned to them.

Applying the scheduling defined in Chapter 5 at the network layer of the wireless hosts leads to:

- In a station, real-time packets are always transmitted first than the elastic packets of the same station.
- The bandwidth received by a station is distributed among the flows of this station proportionally to their weights.

In order to achieve the desired behavior, we require the following conditions in addition to the ones given above:

1. A station that has a real-time packet to transmit should be given a prioritized access to the channel with respect to a station with an elastic packet.

[1]The reference scenario on which the Wireless UFD architecture is based consists of a Wireless LAN which is the user's network and a wired network which is the user's ISP. Note that in this case the entity *user* is most likely an organization such as a company.

2. The bandwidth left by real-time traffic should be shared among the elastic traffic of the different stations according to:

$$\frac{r_s}{\sum_{i \in S} W_i} = \frac{r_r}{\sum_{i \in R} W_i} \; \forall r, s \tag{7.1}$$

   where S is the set of flows of station s and R the set of flows of station r.

In order to satisfy conditions 1 and 2, the MAC protocol of 802.11 has to be modified. In the following we describe two modifications, one to ensure condition 1 (real-time traffic extension) and the other to ensure condition 2 (elastic traffic extension).

7.2.1 Real-time traffic extension

The only solution in the current 802.11 MAC protocol that allows a prioritized access is the PCF mode. With PCF, the prioritized access to the medium is achieved by using a shorter IFS. In the Wireless UFD architecture, we redefine the PCF function of the current standard into a distributed scheme to support real-time traffic. We argue that distributed control is more efficient and flexible than centralized control. The original PCF is not widely supported in current products, and the only requirement of our solution is that the original PCF must not be used in a network together with the extension presented here.

Redefining the PCF mode for real-time allows stations with real-time traffic to access the channel for the transmission of their packets after the PIFS, while stations with elastic packets have to wait until the end of the DIFS. In this way, real-time traffic receives a prioritized access over elastic traffic: whenever there is a real-time packet to be transmitted, it is always transmitted before any other packet.

The mechanism explained so far solves the contention between real-time and elastic packets by giving a higher priority to the former. However, different stations with real-time traffic may still collide when trying to access the channel after the PIFS. For this reason, a contention resolution algorithm is needed in order to avoid collisions between stations with real-time traffic. This algorithm is explained in detail in Section 7.3.

In order to meet the requirement of real-time traffic for low delay, the amount of this type of traffic admitted should be kept sufficiently low via admission control. If there is too much real-time traffic, the resolution of contention in the redefined PCF will take too long and the requirement for immediate delivery of real-time packets will not be met. Thus, the admission of a new request in the Wireless UFD architecture consists of two steps:

1. Check that there are enough resources for this request in the wireless part (i.e. the user's Wireless LAN).

2. Check that there are enough resources in the wired part (i.e. the ISP's wired network).

Thus, a new request will only be accepted if the two conditions above are met. The evaluation of the second condition (the wired part) is performed by the user-based admission control (UBAC) scheme (see Section 5.3), while the first condition (the wireless) is evaluated according to the rule we have obtained via simulation in Section 7.5.1.

7.2.2 Elastic traffic extension

In the DCF mode, the bandwidth received by a station depends on its CW: the smaller the CW, the higher the throughput. In our proposal, elastic traffic is supported by the DCF function of the current standard with minor changes in the computation of the CW in order to give to each station a throughput proportional to the sum of the weights of its flows (Equation 7.1). The algorithm for computing the CW for elastic traffic is explained in detail in Section 7.4.

7.2.3 Protocol Operation

The combination of the mechanisms for real-time and elastic traffic explained above lead to the protocol operation shown in the example of Figure 7.2.

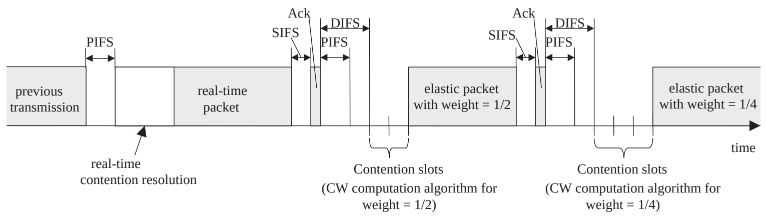
PIFS
Ack
SIFS
DIFS
PIFS
Ack
SIFS
DIFS
PIFS
previous transmission
real-time packet
elastic packet with weight = 1/2
elastic packet with weight = 1/4
time
real-time contention resolution
Contention slots (CW computation algorithm for weight = 1/2)
Contention slots (CW computation algorithm for weight = 1/4)

Figure 7.2: Protocol Operation.

In this example, after the end of a previous transmission, one station has a real-time packet to transmit. It accesses the channel, at the end of the PIFS. In order to make sure that collisions with other stations accessing the channel for real-time traffic are resolved, an additional contention resolution scheme is applied. After the end of the transmission, the receiver answers with an acknowledgement after a SIFS.

In the next access cycle, there is no real-time traffic to be transmitted, so the channel can be accessed by elastic traffic. In the example, it is station with a flow that has a weight of 1/2 that accesses the channel. The packet waits for the end of the DIFS and another two contention slots before it starts its transmission. As commented before, this station fully complies with the existing DCF MAC scheme, but has a different CW, according to the weight of its flow. The receiver again answers with an ACK. Finally, a station with a flow that has a weight of 1/4 accesses the channel with a larger CW.

7.3 Contention Resolution Algorithm for the Real-time Traffic extension (CRA-RT)

The principle of the CRA-RT scheme is shown in Figure 7.3. This scheme uses principles of the EY-NPMA MAC protocol described in [85], and, according to [87], has a residual collision rate almost independent from the number of contending stations. However, the parameters of the contention resolution (CR) scheme as used in [85] will be adapted to the requirements of the CRA-RT scheme.

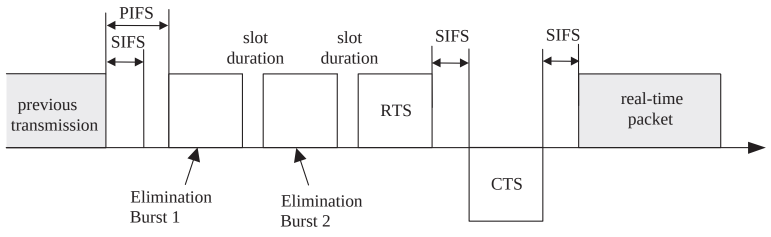

Figure 7.3: Contention resolution scheme for real-time traffic.

A station with real-time traffic starts its contention cycle when a PIFS has passed after the end of a previous transmission. CRA-RT uses two bursts for elimination, elimination burst (EB) 1 and EB2. These bursts may consist of a random data or pseudo-noise pattern. Their only purpose is to occupy the channel such that stations with elastic traffic cannot sense the channel idle for longer than a PIFS duration and, hence, do not interfere an access attempt for real-time.

The duration of the EBs are multiples of the Slot Duration defined in the 802.11 standard. The duration of EB1 is calculated according to the following probability density:

$$P_{E1}(n) = \begin{cases} p_{E1}^{n-1}(1 - p_{E1}) & ; 1 \leq n < m_{E1} \\ \\ p_{E1}^{m_{E1}-1} & ; n = m_{E1}, \end{cases} \tag{7.2}$$

where n is the number of slot durations EB1 shall last, p_{E1} is a probability parameter between 0 and 1 and m_{E1} is the maximum number of EB1 slots. Note that the above formula requires that EB1 lasts at least one slot. This is necessary in order to occupy the channel and keep terminals with elastic traffic from making an access attempt.

The duration of EB2 shall be calculated according to the probability density

$$P_{E2}(n) = \frac{1}{m_{E2}} \quad \text{for } 1 \leq n \leq m_{E2}, \tag{7.3}$$

i.e. it is taken from an equally distributed variable in the range between 1 and the maximum number of EB2 slots, m_{E2}. Note that here the duration is at least one slot for the same reasons as for EB1.

A station that makes an access attempt, first chooses the duration of EB1 and EB2. If it senses the channel free for at least a PIFS, it transmits its EB1. After this transmission, the station senses the channel for one slot duration. If the channel is sensed free, it continues to send its EB2 after the sensing slot. After the transmission of EB2, it senses the channel again. If it is free, it starts to transmit its RTS after a slot duration and the transmission continues as defined for the data transmission using the DCF. If, however, the station senses the channel busy after its transmission of EB1 or EB2, it withdraws its transmission attempt and defers until the channel has been free for at least a PIFS. Using this mechanism, the station which chooses the longest EB1 and EB2 among all contending stations wins the contention and is allowed to transmit.

If two stations happen to have the same EB1 and EB2 durations, they collide. However, due to the importance of the packets, we use the already defined mechanisms in 802.11 for collision detection, i.e. the RTS/CTS handshake and the transmission of an Ack after the packet reception. In fact, the Ack will be transmitted in any case if a packet is being transmitted from a station using the new scheme to a station using the old scheme and, hence, shall be kept for the sake of backwards compatibility.

The CRA-RT scheme will be analyzed mathematically in the following section. It will be shown that the performance of the scheme depends on the values of p_{E1}, m_{E1} and m_{E2}, but also on the number of stations entering a contention cycle. The goal of the mathematical analysis is to parameterize the CRA-RT scheme appropriately for the application in the IEEE 802.11 extension. It will also be a basis to justify the selection of this specific scheme, even if there might have been other possible candidates.

7.3.1 Mathematical Analysis

The idea for the analysis has been taken from [87], although the results differ significantly.

Assume that N_1 stations enter the EB1 period of a specific CRA-RT contention resolution cycle where the duration of the EB1 of each station is given according to Equation 7.2. Then, the probability that the EB1 period ends after i slots and exactly k stations survive, is given by:

$$P_{E1,i,k}(i,k) = \begin{cases} (1-p_{E1})^k & ; i=1;\ k=N_1 \\ \binom{N_1}{k}\left[p_{E1}^{i-1}(1-p_{E1})\right]^k \cdot \left(1-p_{E1}^{i-1}\right)^{N_1-k} & ; 1<i<m_{E1} \\ \binom{N_1}{k}\left(p_{E1}^{i-1}\right)^k \cdot \left(1-p_{E1}^{i-1}\right)^{N_1-k} & ; i=m_{E1} \end{cases} \tag{7.4}$$

Consequently, the probability that exactly k stations survive EB1, can be represented as:

$$P_{E1,k}(k) = \begin{cases} \sum_{i=2}^{m_{E1}} P_{E1,i,k}(i,k) & ; 1 \le k < N_1 \\ \sum_{i=1}^{m_{E1}} P_{E1,i,k}(i,k) & ; k = N_1 \end{cases} \tag{7.5}$$

These k stations are the ones that enter the EB2 period. The average duration $\bar{T}_{E1}$ of EB1 can be calculated as:

$$\bar{T}_{E1} = \mathrm{E}\left(P_{E1,i}(i)\right) \cdot T_{slot} = T_{slot} \cdot \sum_{j=1}^{m_{E1}} j \cdot P_{E1,i}(j) \tag{7.6}$$

where $\mathrm{E}(\cdot)$ denotes the expected value, T_{slot} a slot duration in IEEE 802.11 and

$$P_{E1,i}(i) = \begin{cases} (1-p_{E1})^{N_1} & ; i=1 \\ \sum_{k=1}^{N_1} P_{E1,i,k}(i,k) & ; 1 < i \le m_{E1} \end{cases} \tag{7.7}$$

is the probability that the EB1 period ends after i slots.

The same calculation is now being performed for the EB2 cycle, cf. Equation 7.3. Let N_2 denote the number of stations entering the EB2 cycle. Then, the probability that EB2 ends after i slots with k stations left, is given by:

$$P_{E2,i,k}(i,k,N_2) = \begin{cases} \left(\frac{1}{m_{E2}}\right)^k & ; i = 1;\ k = N_2 \\ \binom{N_2}{k} \frac{(i-1)^{N_2-k}}{m_{E2}^{N_2}} & ; 1 < i \leq m_{E2} \end{cases} \tag{7.8}$$

The expected duration of an EB2 cycle depends on the outcome of the EB1 cycle in terms of numbers of surviving stations and can be represented as:

$$\begin{aligned} \bar{T}_{E2}(N_2) &= T_{slot} \cdot \sum_{N_2=1}^{N_1} P_{E_1,k}(N_2) \cdot \mathrm{E}\left(P_{E_2,i}(i,N_2)\right) \\ &= T_{slot} \cdot \sum_{N_2=1}^{N_1} \left[P_{ES,k}(N_2) \cdot \sum_{i=1}^{m_{E2}} \cdot P_{E2,i}(i,N_2) \right] \end{aligned} \tag{7.9}$$

where

$$P_{E2,i}(i,N_2) = \begin{cases} \left(\frac{1}{m_{E2}}\right)^k & ; i = 1;\ k = N_2 \\ \sum_{k=1}^{N_2} P_{E2,i,k}(i,k,N_2) & ; 1 < i \leq m_{E2} \end{cases} \tag{7.10}$$

denotes the probability that the EB2 cycle ends after i slots. The overall collision probability P_c is the situation where more than one station survive the EB2 cycle and can be calculated as:

$$P_c = \sum_{N_2=2}^{N_1} \left(1 - P_{E_2,k}(1,N_2)\right) \cdot P_{E1,k}(N_2) \tag{7.11}$$

with

$$P_{E_2,k}(k,N_2) = \begin{cases} \sum_{i=2}^{m_{E2}} \binom{N_2}{k} \frac{(i-1)^{N_2-k}}{m_{E2}^{N_2}} & ; 1 \leq k < N_2 \\ \left(\frac{1}{m_{E2}}\right)^{N_2} + \sum_{i=2}^{m_{E2}} \binom{N_2}{k} \frac{(i-1)^{N_2-k}}{m_{E2}^{N_2}} & ; k = N_2 \end{cases} \tag{7.12}$$

as the probability that k out of N_2 stations survive the EB2 cycle.

The overhead O_1 of a single access attempt depends on three main values:

- The expected duration T_{CRA-RT} of a successful CRA-RT cycle, i.e. a cycle which finishes without a collision. This is given by the sum of $\bar{T}_{E1}$, see Equation 7.6, $\bar{T}_{E2}$, see Equation 7.9, and $2 \cdot T_{slot}$ for the carrier sensing slots after EB1 and EB2.
- The time it takes to detect a collision of the CRA-RT scheme, T_{coll}. A collision will be detected if the RTS of the sender is not answered by a CTS of the receiver. The medium can be accessed again by a real-time station after a PIFS following the RTS. This time is denoted by T_{RTS}, and $T_{coll} = T_{CRA-RT} + T_{RTS}$.
- The collision probability P_c according to Equation 7.11.

The overhead for a single access attempt in terms of average duration of the CRA-RT scheme, then, is calculated as:

$$O_1(m_{E1}, p_{E1}, m_{E2}, N_1) = P_c \cdot T_{coll} + (1 - P_c) \cdot \bar{T}_{CRA-RT} \tag{7.13}$$

Iterating this overhead of a single access cycle for subsequent access cycles, weighted with the residual collision probability for a single attempt, yields the average overhead O:

$$O(m_{E1}, p_{E1}, m_{E2}, N_1) = O_1 \cdot \frac{1}{1 - P_c} \tag{7.14}$$

The overhead O can be interpreted as a function that weighs the overhead of the collision avoidance scheme against the additional overhead that needs to be spent if collisions occur. It is clear that, the more overhead is spent for the collision avoidance, the smaller the collision probability. On the other hand, each collision adds a certain well-known amount of overhead because it can be detected due to the RTS/CTS scheme used. Note that O depends on the parameters given in Equation 7.14. The optimum parameter set for m_{E1}, p_{E1} and m_{E2} is found when the overhead O reaches its minimum for a given N_1, i.e. we seek $\min(O(m_{E1}, p_{E1}, m_{E2}, N_1)$. The function O has been computed and has the following properties:

- There is always a dedicated minimum for a given value of N_1.
- The minimum is very stable for values of m_{E1} and m_{E2} bigger than the ones for the minimum.
- The value of p_{E1} can be chosen from a big range around the optimum value without significant impact on the overhead.

- The bigger N_1, the bigger the value of the optimum value for p_{E1}. The optimum values for m_{E1} and p_{E1} remain almost unchanged.

- The residual collision probability P_c decreases with increasing values of m_{E1}, p_{E1} and m_{E2}. The increase of m_{E2} has the biggest impact.

The selection of N_1 depends on the usage scenario. For the optimization of the CRA-RT scheme, it was assumed that 10 real-time stations almost completely occupy the available data rate of the Basic Service Set. This scenario was assumed to be the worst case for the 2 Mbit/s DS modus of IEEE 802.11 and was chosen as the reference scenario. By iteratively performing simulations and adapting m_{E1}, p_{E1} and m_{E2}, it was found that in this scenario, on average approximately seven stations enter each CRA-RT cycle. Therefore, $N_1 = 7$ was chosen. The resulting optimum values for m_{E1}, p_{E1}, m_{E2}, the resulting overhead O and the residual collision probability P_c are:

$$\begin{aligned}
p_{E1} &= 0.43\\
m_{E1} &= 5\\
m_{E2} &= 4\\
O &= 218.69\mu s\\
P_c &= 10.4\%
\end{aligned} \tag{7.15}$$

All simulations presented in Section 7.5.1 have been performed using this parameter set.

7.3.2 Rationale for choosing the CRA-RT scheme

The CRA-RT scheme can be compared to other schemes with only one elimination phase. The bursting is necessary because of the carrier sensing of legacy stations that may interrupt a CRA-RT cycle. Assume the following very simple scheme with similar overall overhead as the CRA-RT scheme with the parameters according to Equation 7.15: Each station chooses a random number of slots for its elimination burst duration out of 9 possible slots. After the bursting, the stations sense the channel for 1 slot. If it is free, they immediately continue with the transmission of an RTS, otherwise they withdraw their access attempt. Assuming an equal distribution according to Equation 7.3, the overhead can be calculated according to the equations given above for the EB2 cycle. The overhead O for the reference scenario, then, has a value of approx. $226\mu s$ at a residual collision rate of $P_c = 34.6\%$. It is immediately obvious that the overhead is bigger and! the number of access attempts for a single packet to be transmitted is higher than with the

CRA-RT scheme. A similar calculation with similar results can be performed for a geometric distribution.

It is the combination of the two EB cycles with the probability distributions according to Equations 7.2 and 7.3 which makes the CRA-RT scheme very efficient. The EB1 cycle has the property that the probability for a low number of surviving stations entering the EB2 cycle, is high, almost independent from the number N_1 of contending stations. The EB2 cycle, then, is well suited to sort out a single station out of a low number N_2 of remaining stations.

7.4 Contention Window Computation for the Elastic Traffic extension

In the DCF mode of the 802.11 standard, the size of the CW determines the probability for a station to win the contention. The smaller the CW is, the higher the probability of getting access to the channel. As a consequence, there is a direct relationship between the CW assigned to a station and the bandwidth that this station will receive in a specific scenario. Equation 7.1 can therefore be satisfied by assigning to each station the CW values that lead to the desired bandwidth distribution for elastic traffic.

The difficulty of this approach, however, relies in determining the CW that will lead to the specified distribution. The approach we have chosen for the calculation of the CW in Wireless UFD is a dynamic one.

In order to be able to properly adjust the CWs, we introduce a new variable L_s^{MAC}, the *MAC label*[2], defined as:

$$L_s^{MAC} = \frac{r_s}{\sum_{i \in S} W_i} \tag{7.16}$$

where r_s is the estimated bandwidth experienced by station s, S is its set of flows and W_i their weight.

With the above definition of L_s^{MAC}, the resource distribution expressed in Equation 7.1 can be achieved by imposing the condition that the MAC label L_s^{MAC} should have the same value for all the stations:

$$L_s^{MAC} = L \quad \forall s \tag{7.17}$$

[2]The MAC label defined in this chapter should not be confused with the packet label L_k of the previous chapters. The two labels serve different purposes, are computed independently and travel in different parts of the packet header: while the former is inserted in the MAC header, the latter travels in the network header.

Note that the actual value of L can vary in time (depending on the number of flows for example).

Equation 7.17 is fulfilled by using the following algorithm: having calculated its own L_s^{MAC}, each station includes it in the MAC header of the packets it sends. For each observed packet, if the L_s^{MAC} in the packet's MAC header is smaller than the L_s^{MAC} of the station, the station increases its CW by a small amount, while in the opposite case the station decreases its CW by a small amount. In this way, the L_s^{MAC} of all the stations tend towards a common value, L.

The above explanation describes the basics of the algorithm. However, in the adjustment of the CW, there are additional aspects that have to be taken into account:

- For backward compatibility reasons, we do not want the CW to increase above the values defined by the 802.11 standard, since this would lead to Wireless UFD stations experiencing a worse performance than 802.11 legacy terminals when competing with them in the same Wireless LAN.
- If the low sending rate of the application is the reason for transmitting below the desired rate, then the CW should obviously not be decreased. This can be detected by the fact that in this situation the transmission queue is empty.
- CWs should not be allowed to decrease in such a way that they negatively influence the overall performance of the network. If the channel is detected to be below its optimum limit of throughput due to too small values for the CWs (i.e. *overload*), the CW should be increased. This aspect will be elaborated in the following section.

The above considerations lead to the Algorithm 4. This algorithm computes a value p which is used to scale the CW values defined in 802.11. Note that, besides this scaling of the CW, the backoff time computation algorithm is left as defined in the 802.11 standard (i.e. the Contention Window is doubled every unsuccessful transmission attempt for a given number of times).

7.4.1 Overload

Algorithm 4 does not consider one important issue which is the *overload.* In fact, due to the nature of our protocol and in particular due to the dynamic way of adjustment of the size of the CW, a mechanism for controlling the *overload* is necessary. As we can see in Algorithm 4, each station adjusts

Algorithm 4 CW computation.

```
For each observed packet:
$L_{own}^{MAC}$ = get_MAC_label_station()
$L_{rcv}^{MAC}$ = get_MAC_label(packet)
$\Delta_1 = k\left|\frac{L_{own}^{MAC}-L_{rcv}^{MAC}}{L_{own}^{MAC}+L_{rcv}^{MAC}}\right|$
if $L_{own}^{MAC} > L_{rcv}^{MAC}$ then
  $p = (1+\Delta_1)p$
else
  if queue_empty then
    $p = (1+\Delta_1)p$
  else
    $p = (1-\Delta_1)p$
  end if
end if
$p = min\{p, 1\}$
$CW = p \cdot CW_{802.11}$
```

its CW only on the basis of its own requirements. Such "selfishness" can easily be disastrous, due to the following side effect of the small CWs. We have been arguing so far that, the smaller the CW for a given station, the larger the throughput received by this station. The other, bad consequence of such a procedure is that the more stations have small CWs, the bigger the probability of a collision. One can easily see that, for a big number of stations with flows of high weight, this can lead to an absolute blockage of the channel. Once all of the stations start decreasing their CWs in order to get the desired relative throughput, the number of! collisions will start increasing, leading to even smaller CWs, and as a consequence, continuous collisions. A solution to this problem is to extend Algorithm 4 with an overload condition, as shown in Algorithm 5.

Let us now explain how we actually detect *overload*. As we have mentioned before, a big number of stations trying to transmit with high weight flows, i.e. decreasing their CWs, leads to an increase of the number of collisions. If we now provide each station with a collision counter[3], which determines how many collisions a packet experiences before it is successfully transmitted, we can write the following simple condition determining over-

[3]Note that in 802.11 collisions can only be detected through the lack of the Ack. However, a missing Ack can also be caused by other reasons different than a collision. In Section 7.5.2 we study the impact into our algorithm of having missing Ack due to errors in the channel.

Algorithm 5 CW Computation with overload avoidance.

```
For each observed packet k:
if overload then
  $p = (1 + \Delta_2)p$
else
  if $L_{own}^{MAC} > L_{rcv}^{MAC}$ then
    $p = (1 + \Delta_1)p$
  else
    if queue_empty then
      $p = (1 + \Delta_1)p$
    else
      $p = (1 - \Delta_1)p$
    end if
  end if
end if
$p = min\{p, 1\}$
$CW = p \cdot CW_{802.11}$
```

load

$$\texttt{if}\ \ (av\_nr\_coll > c)\ \ \texttt{then}\ \ overload = true \tag{7.18}$$

where c is a constant that has to be properly adjusted. If c is too low, high priority stations (i.e. stations with flows of high weight) will not be allowed to decrease their CWs sufficiently, and as a consequence they will not be able to achieve the desired differentiation. On the other hand, if c is too large, the number of collisions in the channel will be very high and the overall performance will be harmed. This constant, therefore, represents a tradeoff between the level of differentiation between high and low priority stations and the efficiency (i.e. total throughput) of the channel. These tradeoff has been studied via simulation (see Section 7.5.2), and an optimum value for c has been chosen according to simulation results.

The average number of collisions, (av_nr_coll), in Equation 7.18 is calculated after each successful transmission in the following way

$$av\_nr\_coll = (1 - t) * num\_coll + t * av\_nr\_coll \tag{7.19}$$

where in order to smoothen its behavior, we use some sort of memory, taking into account the last calculated value of av_nr_coll (on the rhs of Equation 7.19). The constant t is a small number playing the role of a smoothening factor.

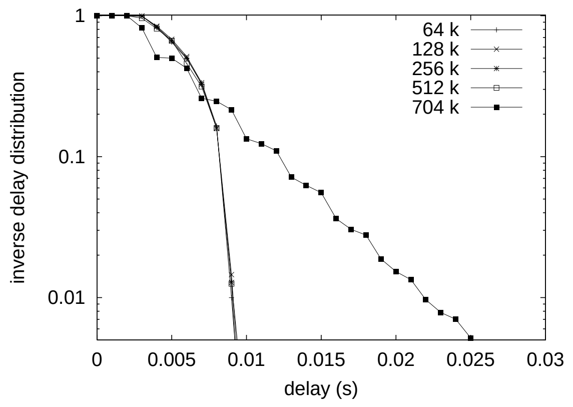

Figure 7.4: Inverse delay distribution for 2 real-time stations, CBR, 500 byte packet length.

7.5 Simulations

To test the performance of the Wireless UFD architecture presented in this chapter, we simulated it on a network consisting of a number of wireless terminals in a 2 Mbps Wireless LAN communicating with a wired node.

7.5.1 Real-time Traffic

For the purpose of simulating the contention resolution scheme described in Section 7.3, the existing implementation of the 802.11 MAC protocol in ns-2 was extended by the functions necessary for the real-time traffic extension. In all simulations, stations sending elastic traffic coexist with stations sending real-time traffic, in such a way that each station sends either real-time or elastic traffic. The elastic traffic stations always have something to transmit. The traffic of the real-time stations is of UDP type, since UDP is usually applied in conjunction with real-time applications.

As a quality criterion, we set a maximum delay of 25 ms. This limit shall not be exceeded by 3% or more of the packets. Therefore, the emphasis of all simulations is on delay. The total number of stations in all simulations is 20, i.e. the number of stations sending elastic traffic is 20 minus the number of real-time stations. The stations are located such that they are all within communication distance of each other.

Simulation results for constant bit rate (CBR) sources with 500 bytes packet length are shown in Figures 7.4 and 7.5. The simulation results in Figure 7.6 are obtained for 100 bytes packet length. Each of them shows an inverse distribution of the delay in seconds for a given number of real-time

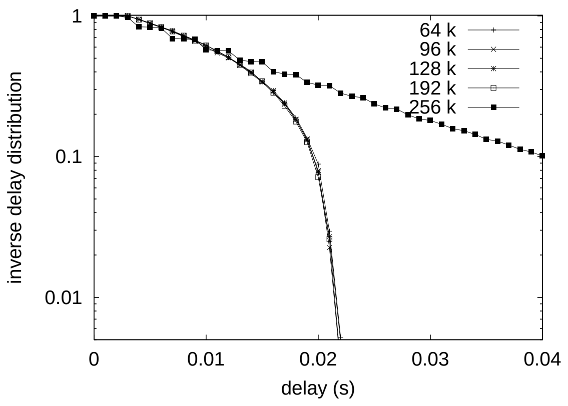

Figure 7.5: Inverse delay distribution for 6 real-time stations, CBR, 500 byte packet length.

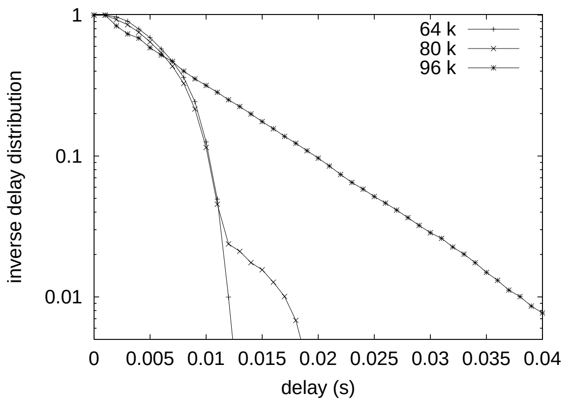

Figure 7.6: Inverse delay distribution for 6 real-time stations, CBR, 100 byte packet length.

stations with the data rate of a single station as parameter. The interpretation of the graphs is as follows: If one is interested to know how many percent of the packets have a delay higher than the one selected, one must pick the time on the x-axis and read the corresponding number on the y-axis. A number of 0.1 on the y-axis means that 10% of the packets were later than the selected time.

The simulations show that the air interface can bear a real-time traffic saturation throughput of about 1300 Kbps for 500 bytes packet length and of below 700 Kbps for a packet length of 100 bytes. In general, the saturation throughput decreases with decreasing packet length because each packet carries a certain, more or less constant overhead. However, it is likely that some real-time applications, such as voice over IP, use short packet lengths and, hence, the performance of the CRA-RT scheme for short packets is important.

As long as the required total data rate of the real-time stations remains below the saturation throughput, the actual throughput of each real-time station corresponds to its required data rate. The data rate left is being used by the elastic traffic stations. If the required data rate of the real-time stations exceeds the saturation throughput, the real-time stations share the maximum data rate equally, whereas the elastic traffic stations get no throughput at all.

Figure 7.4 shows the results for two stations sending real-time traffic. The data rates range from 64 Kbps up to 704 Kbps per real-time station. As can be seen, the delay for all data rates up to 512 Kbps remains below 10 ms in all cases. The data rates achieved by the real-time stations corresponds to the data rate delivered by the traffic sources, i.e. they can deliver all offered packets to the destination. The delay increases at a data rate of 704 Kbps but still remains below the allowed limit. In this case, however, the throughput of the real-time stations is limited to approximately 650 Kbps, i.e. half of the saturation throughput for each station. The elastic traffic stations could not deliver any packet during this simulation run.

The curves depicted in Figure 7.5 show a similar situation. The delays are higher than with two stations but still remain in the allowed region. If each station uses 256 Kbps, the saturation throughput is exceeded and the delays increase significantly. The same scenario with a packet length of 100 bytes per real-time station is shown in Figure 7.6. The quality criterion can be met for 6 stations with 64 Kbps each but the saturation throughput is already reached with a data rate of 96 Kbps per real-time station.

The fact that the delay distribution is almost independent from the data rate of each terminal is quite in line with the results obtained in [87]. An interpretation of the results is that, as long as the real-time stations do not

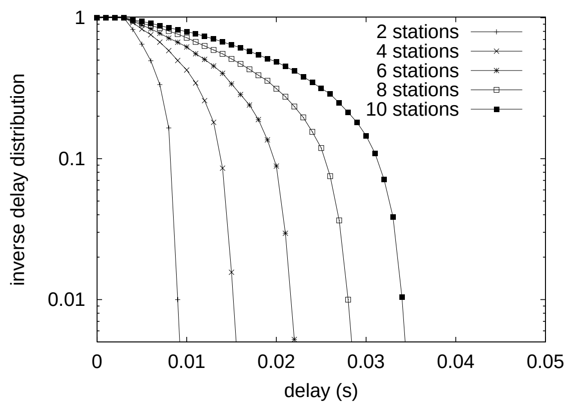

Figure 7.7: Inverse delay distribution 64 Kbps with varying numbers of real-time stations, CBR, 500 bytes packet length.

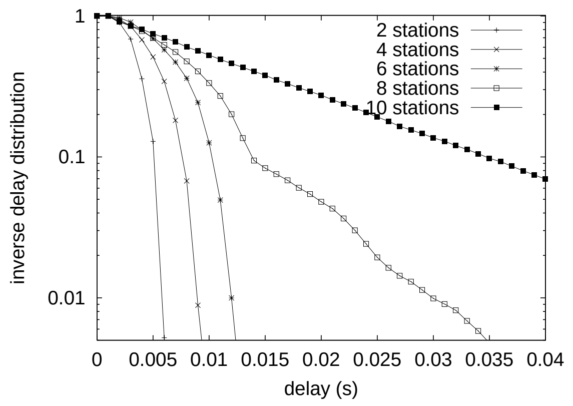

Figure 7.8: Inverse delay distribution 64 Kbps with varying numbers of real-time stations, CBR, 100 bytes packet length.

always have something to transmit, they do almost not interfere with each other and the transmission delay of the packets plus the waiting time for the end of an ongoing transmission is decisive. Since the packet length is constant for all real-time and elastic traffic stations, all have to wait equally long on average. Furthermore, the delay depends on the number of elastic traffic stations contending for the channel. The dependence of the transmission delay on the packet length and on the number of elastic traffic stations is a subject for further study.

The graphs in Figure 7.7 show the delay distributions for a data rate of 64 Kbps per station at a packet length of 500 bytes and varying numbers of stations. The same situation for a packet length of 100 bytes is depicted in Figure 7.7. It can be seen from the graphs that the quality criterion is

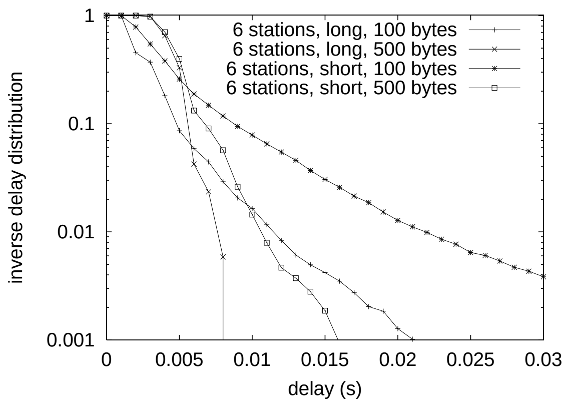

Figure 7.9: Inverse delay distribution 64 Kbps for 6 real-time stations, ON/OFF traffic, 100 and 500 bytes packet lengths.

	data rate (kbps)				
stations	32	64	128	256	512
2	x	x	x	x	x
4	x	x	x	x	
6	x	x			
8	x				
10					

Table 7.1: Overview which configurations meet the quality criterion

met for 6 real-time stations. For 8 real-time stations and more, the quality criterion can not be met. Whereas the curve for 6 stations with 500 bytes packet length decreases very rapidly, the curve for 100 byte packets has a longer tail.

The graph in Figure 7.9 shows the inverse delay distribution for ON/OFF traffic for 6 real-time stations with a data rate of 64 Kbps and 100 and 500 bytes packet lengths. The ON/OFF sources have exponentially distributed burst and idle times. The curves denoted with "long" have an average burst time of 2 s and idle time of 8 s, whereas the curves with "short" have average burst and idle times of 500 ms. The data rate is the average data rate of the real-time stations, i.e. the peak data rate is higher than the average rate. The curves for 500 bytes packet length are rather steep, whereas the 100 bytes curves have a rather long tail. All scenarios meet the quality criterion.

Table 7.1 shows which combinations of numbers of real-time stations and data rate per real-time station meet the quality criterion for all possible scenarios, i.e. for 500 and 100 bytes packet lengths as well as for CBR and

ON/OFF traffic. A cross in the table entry means that the criterion is met. As a rule of thumb, an admission control derived from this table would allow not more than six stations and not more than a data rate of 64 Kbps per station sending real-time traffic. As can be seen from the delay graphs above, a more sophisticated admission control scheme would have to consider additional criteria, such as the packet length and burstiness of real-time applications.

It can be seen from the simulation results that the contention resolution scheme described in Section 7.3 can meet the quality criterion reliably for at least 6 real-time stations with data rates up to 64 Kbps per station for CBR and ON/OFF traffic with packet lengths of 100 and 500 bytes. If the available bandwidth is used by real-time stations up to or close to the saturation throughput, however, the real-time stations use most or all of the available bandwidth and the service quality of elastic traffic stations drops dramatically.

7.5.2 Elastic Traffic

For the simulation of the elastic traffic extension described in Section 7.4, Algorithm 5 was inserted into the existing implementation of the 802.11 MAC DCF protocol in ns-2.

We chose to use the RTS/CTS mechanism in all cases. This mechanism, optional in the 802.11 standard, increases bandwidth efficiency in case of many collisions, since with this mechanism collisions occur with the relative small control packets rather than with long data packets. Since our architecture may lead to larger number of collisions than the normal 802.11 MAC DCF, this mechanism can be especially beneficial in our case.

Unless otherwise specified, we use the following parameters for the simulations: $k = 0.01$, $\Delta_2 = 0.25$ and $t = 0.25$. All stations are sending only elastic traffic and just one flow. We refer to the weight of this flow as the station's weight.

Instantaneous Bandwidth Distribution

In Wireless UFD the desired bandwidth distribution for elastic traffic is achieved by adjusting adaptively the CW of elastic traffic stations according to their measured bandwidth. Figure 7.10 shows this dynamic adjustment; the simulations correspond to a scenario with a total number of 10 stations, 2 of them with a weight of $2W$ and the rest with a weight of W. All stations are sending UDP CBR traffic with a packet size of 1000 bytes. It can be seen

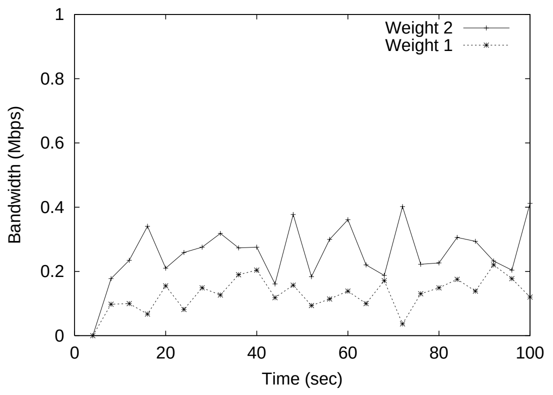

Figure 7.10: Instantaneous Share for Elastic traffic.

when comparing the instantaneous bandwidth of a high priority and a low priority station that their ratio oscillates around the desired value (i.e. 2).

The value of W in the above weights is such that the sum of the weights of all the flows of the user equals 1, in concordance with the definition of weight W_i in Chapter 2 (Equation 2.12). Actually, the specific value of W has no meaning in this chapter, since the absolute value of the weights has no impact on the distribution of the Wireless LAN capacity. For the sake of simplifying the notation, hereafter we will only indicate the relative values of the weights when describing the simulation scenarios (i.e. we will refer to a weight of 1 for expressing a weight equal to W, and to a weight of 2 for expressing a weight equal to $2W$).

Bandwidth Distribution as a function of the weight

With Wireless UFD, the throughput experienced by a station should be proportional to the weight assigned to its flow. Figure 7.11 shows the ratio between the throughput experienced by high priority (HP) and low priority (LP) stations (we refer to this ratio as the *experienced weight*) as a function of the weight assigned to the high priority stations when low priority stations have a weight equal to 1. Ideally, the resulting function should be the identity (i.e. a diagonal); that is, an *experienced weight* equal to the weight. In the figure it can be seen that the simulated results are quite close to the ideal. Only in the case of large weights and a large number of stations, the results obtained differ noticeably from the ideal case; however, not even in this case differences are too big (e.g. with 50 stations and a weight of 10, the *experienced weight* is 8).

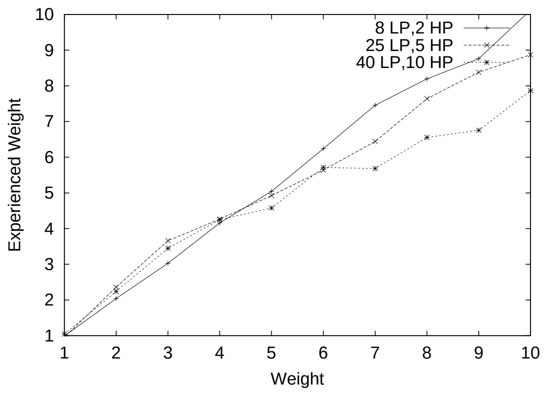

Figure 7.11: Bandwidth Distribution as a function of the weight.

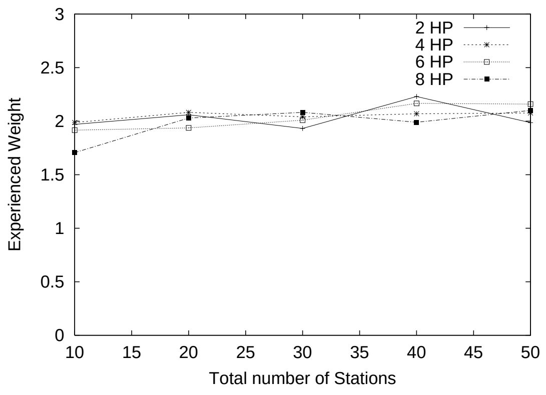

Figure 7.12: Bandwidth Distribution as a function of the number of stations.

Impact of the number of stations

The proposed algorithm for computing the CW in Wireless UFD relies on the experienced throughput estimated by each station. Note that the higher the number of stations, the lower the throughput received by each station. Since a low throughput is more difficult to estimate with exactitude than a high throughput, a large number of stations may negatively impact the performance of the algorithm. Figure 7.12 shows this impact when high priority stations have a weight of 2 and low priority ones have a weight of 1. Note that, in all cases, the experienced ratio between throughput of high and low priority stations keeps close to the desired value, which is 2. We conclude that the number of stations has a negligible impact on the *experienced weight*.

Impact of the parameter c

In Section 7.4.1 the constant c has been defined as the maximum average number of collisions allowed. This limit is needed in order to avoid loss of efficiency due to too small CWs.

Since we are using the RTS/CTS mechanism, the number of collisions will never be bigger than 8 (according to the standard, a packet is dropped after 8 RTS tries). Therefore, the chosen value for c must be in the range of $0 < c < 8$.

In order to analyze the impact of c we chose to use a scenario with a large number of stations (100 stations), half of them with very high weights ($weight = 6$). This scenario leads to many stations with very small CW, and, therefore a high number of collisions, in such a way that collisions are controlled by the parameter c. Note that in a scenario without many collisions the impact of c would be almost null.

Figures 7.13, 7.14 and 7.15 show the total throughput, the number of drops per successful packet and the *experienced weight* as a function of c in the above scenario. In these figures it can be seen that if the value of c is too high, the total throughput experienced is very low, and the percentage of losses very high. In the extreme case ($c > 7$) the throughput drops to 0 and the drops increase drastically. The reason for this is that with such values of c, CWs are allowed to decrease too much and the probability of collision gets too big. Note that in this case low priority stations totally starve and the differentiation tends to infinite, since the whole bandwidth is used by high priority stations.

On the other hand, if the value of c is too low, we obtain a good total throughput and very low losses, but we do not achieve the desired differentiation. In the limit ($c = 0$) there is no differentiation at all and high priority stations get exactly the same throughput as low priority ones (i.e. *experienced weight* $= 1$). The reason for this is that, with such values of c, CWs are not allowed to decrease below the values defined in the 802.11 standard, and, therefore, the elastic traffic extension defined in Section 7.4 is deactivated.

As a conclusion, c expresses a tradeoff between efficiency and differentiation, and it can be adjusted via administration depending on specific user preferences. In the simulations of this section we have chosen to use an intermediate value: $c = 5$. With this value of c, a good level of differentiation is achieved while conserving a good overall efficiency.

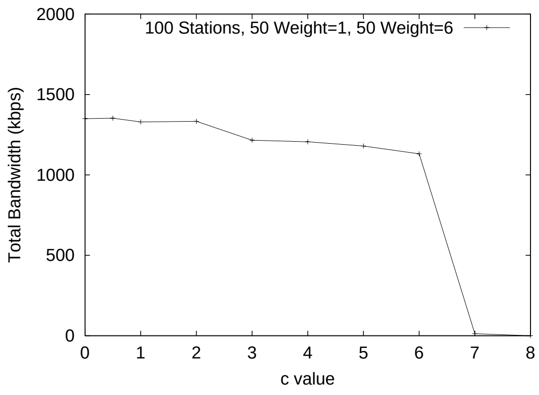

Figure 7.13: Throughput as a function of c.

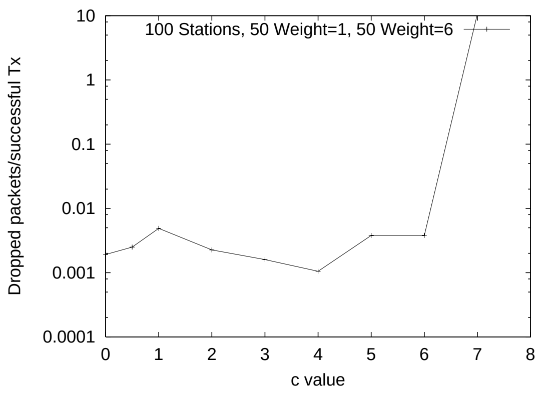

Figure 7.14: Drops as a function of c.

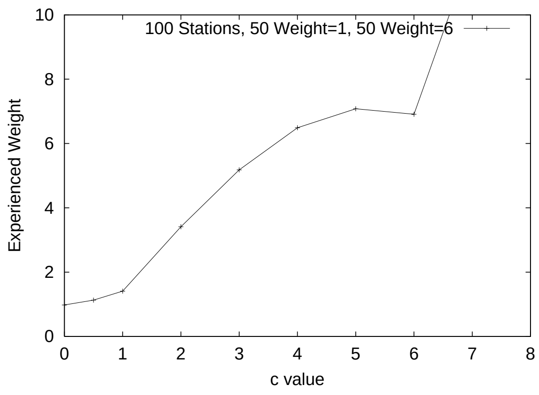

Figure 7.15: Experienced weight as a function of c.

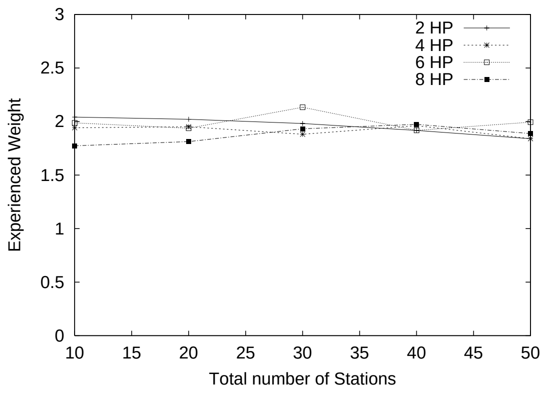

Figure 7.16: Impact of 802.11 terminals.

Backwards Compatibility

Legacy 802.11 stations do not carry in the MAC header the L_s^{MAC} field, and this can have an impact into the overall performance of the Wireless UFD architecture. This impact is studied in the simulation results shown in Figure 7.16. In this simulation, the number of stations with the Wireless UFD architecture is kept to 10, and the rest of the stations are legacy 802.11 terminals. The figure shows the ratio between the throughput of high priority stations (weight 2) and low priority stations (weight 1) as the number of 802.11 terminals increases. It can be seen that this ratio is very close to the desired value, independent of the number of 802.11 terminals.

Channel utilization

Having Wireless UFD stations with a CW smaller than the CW defined in the current standard can impact the channel utilization. Figure 7.17 shows the channel utilization in the same scenario than the described for Figure 7.12, and compares it to the channel utilization with the current standard. It can be seen that the channel utilization keeps always close to the channel utilization of the current standard.

Packet drops

The algorithm proposed in this chapter for elastic traffic increases the aggressiveness of the MAC protocol, since it makes the CW smaller with respect to the current standard. This has an impact on the packet drops experienced at the MAC level, since after a certain number of unsuccessful retries the MAC protocol decides to drop the packet. The more aggressive we are, the

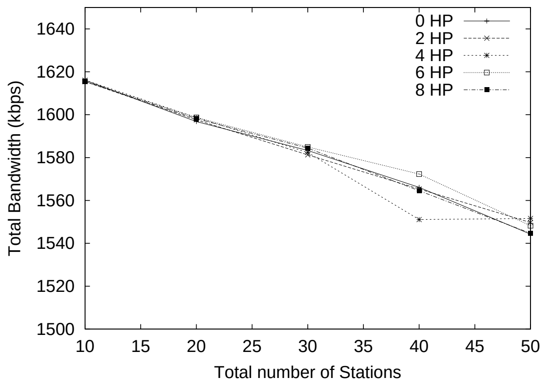

Figure 7.17: Channel utilization.

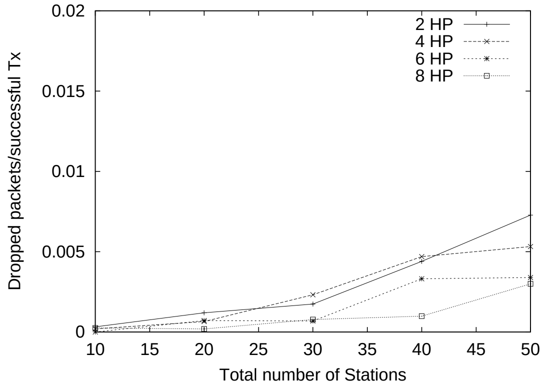

Figure 7.18: Packet drops.

higher is the probability for a retry to fail and, therefore, the probability of experiencing a packet drop.

We studied this impact in the simulations shown in Figure 7.18. It can be seen that packet drops at the MAC level increase with the total number of stations and decrease with the number of high priority stations. This is because the CW required to achieve the desired differentiation for a small number of high priority stations is very low, and therefore the probability of having 8 RTS/CTS collisions (i.e. a packet lost) is higher. However, the number of dropped packets always keeps very low: even in the worst case the percentage of packet drops is below 1%.

Channel Errors

Considering a non-ideal channel a not received Ack can be due to a channel error. As discussed in Section 7.4.1 we have introduced a collision counter

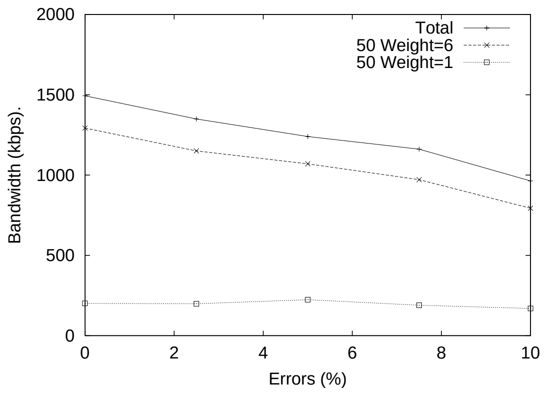

Figure 7.19: Level of differentiation as a function of the error rate

which counts as a collision every sent packet (RTS) for which an Ack (CTS) has not been received. The effect of the channel errors in the collision counter would be the interpretation of a channel error as a collision. This could lead to assume falsely overload in the channel due to channel errors. The result would be an unnecessary increase of the CW, leading to a lower level of differentiation.

We have studied this impact under an extreme scenario as in Figures 7.13 to 7.15 with a value of c equal to 5. We can observe from Figure 7.19 that the level of differentiation is affected by the percentage of errors, as expected. Note that even in an extreme scenario with a high error percentage (10%) we still keep a reasonably high level of differentiation.

Hidden node impact

In order to study the impact of hidden nodes in Wireless UFD we simulated the scenario depicted in Figure 7.20. This scenario consists of three groups of stations (1,2,3) within the coverage area of the Access Point (AP). Group 1 consists of one high priority station with weight equal to 2, and groups 2 and 3 consists of $50 - x - 1$ and x stations, respectively, all with a weight equal to 1. Groups 1 and 3 are hidden from each other.

Note that, since we are using the RTS/CTS mechanism, collisions of data packets due to the hidden node problem are avoided. However, the higher the number of stations hidden from the high priority station, the less accurate the computation of the CW in the high priority station will be. Figure 7.21 shows how this problem impacts the desired bandwidth distribution: the level of differentiation (*experienced weight*) decreases with the number of hidden stations. However, even in the extreme case when 80% of the stations are

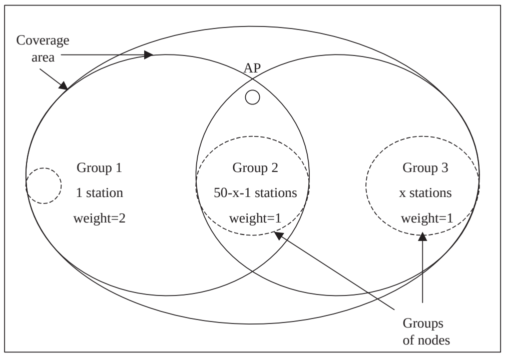

Figure 7.20: Simulation scenario

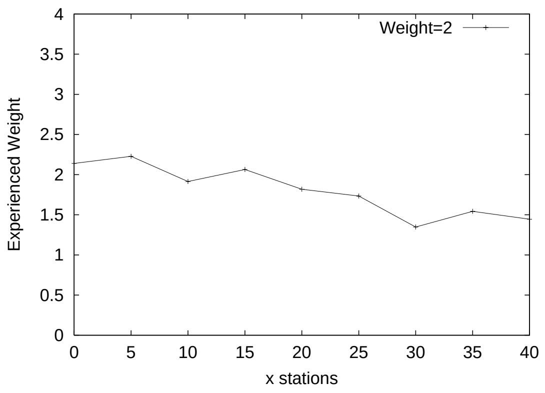

Figure 7.21: Hidden node

hidden ($x = 40$), we still keep a significant level of differentiation.

Impact of bursty traffic

The simulations shown so far correspond to a constant traffic (UDP CBR sources). In order to gain a better understanding of the impact of different traffic sources to the performance of the elastic traffic extension, we simulated it under bursty traffic (UDP ON/OFF sources).

In order to show the impact of different burst sizes, we performed two different simulations: one with a small burst (ON/OFF periods of 1 ms in average), and one with large bursts (ON/OFF periods of 500 ms in average). The simulation scenario was the same as the described for Figure 7.12.

Figure 7.22 shows the results when the ON/OFF periods are of 1 ms. Note that these results are very similar to the results of Figure 7.12 (CBR traffic), which means that short ON/OFF periods do not impact the performance of a station. In Figure 7.23 it can be seen that the results for large ON/OFF

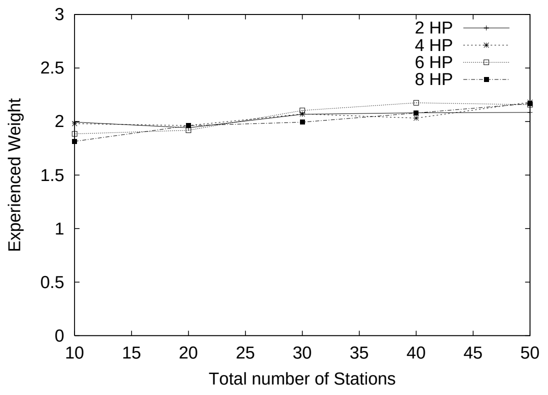

Figure 7.22: Sources UDP ON/OFF 1 ms.

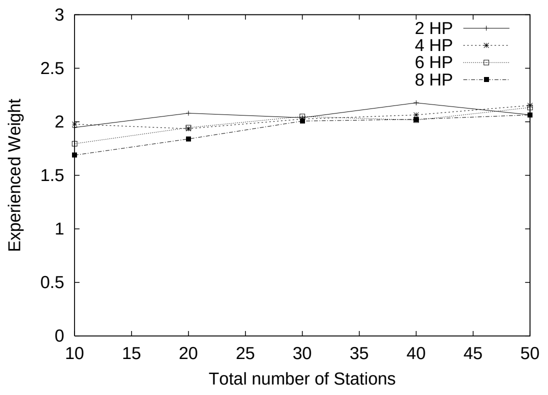

Figure 7.23: Sources UDP ON/OFF 500 ms.

periods are also very similar to the results of Figure 7.12, with a slightly higher oscillation.

TCP sources

Figure 7.24 shows the *experienced weight* of high priority stations for the scenario of Figure 7.12 with TCP sources. It can be seen that there are quite high oscillations, specially when the number of stations is high. This oscillation is due to the congestion control algorithm used in TCP. However, in average, the results obtained tend to the desired ones.

Note that, in contrast to the previous experiments, in this case we have downlink traffic consisting of TCP acknowledgments. Since this traffic consists of several flows, we have multiple flows at the AP. We used the UFD scheduling at the network level of the AP to handle this case. We assigned

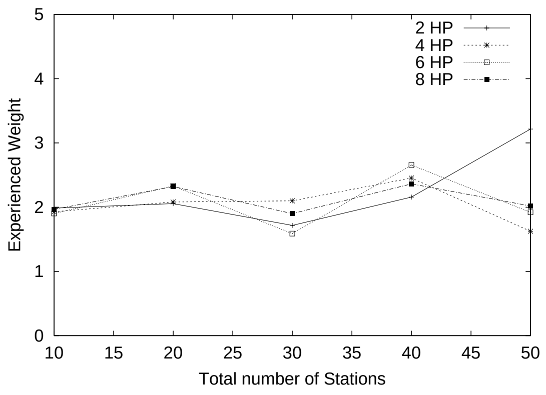

Figure 7.24: TCP Sources.

to the flow i of TCP acknowledgments the same weight as the corresponding flow of TCP data packets. This is necessary in order to achieve the desired bandwidth distribution, since the TCP acknowledgments also impact the throughput of a TCP connection through the congestion control of TCP.

TCP vs UDP

When TCP and UDP flows compete with each other, the bandwidth distribution tends to favor UDP. This is because, in case of congestion, TCP backs off because of its congestion control mechanism, and UDP, without any kind of congestion control and therefore more aggressive, consumes the bandwidth left by TCP. An architecture for bandwidth allocation should overcome this different level of aggressiveness of the sources and provide all sources with their fair share of bandwidth independent of the congestion control algorithm they use.

To study the level of fairness between TCP and UDP achieved by Wireless UFD, we performed the following experiment: two high priority stations had a weight of 2, one sending an endless TCP flow and the other a UDP CBR flow. The remaining 8 stations had a weight of 1 (low priority) and were all sending UDP CBR traffic. Figure 7.25 shows the instantaneous bandwidth achieved by the TCP and UDP high priority sources and one UDP low priority source. It can be seen that, in average, the resulting bandwidth distribution is the desired.

¿From this experiment we conclude that Wireless UFD provides TCP with a fair treatment with respect to UDP. This is because the CW computation algorithm adapts the CW to the aggressiveness of the source: a less aggressive source, like TCP, will see its CW reduced until it receives the

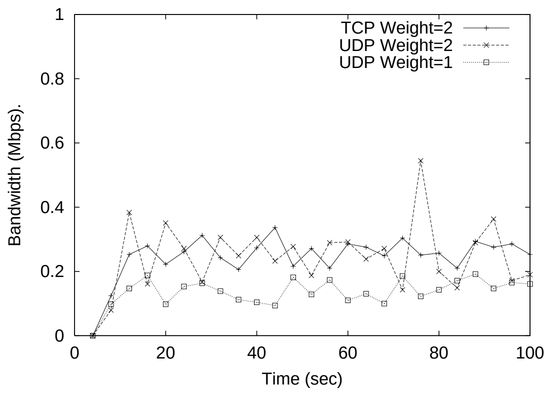

Figure 7.25: TCP vs UDP

desired relative throughput, while a more aggressive source, like UDP, will achieve its desired relative throughput with a larger CW.

7.6 Summary

In this chapter we have proposed the Wireless UFD architecture for providing UFD's resource allocation in Wireless LAN. Wireless UFD consists of two independent extensions to the IEEE 802.11 standard: the extension for real-time and the extension for elastic traffic.

The real-time extension reuses the PIFS of the 802.11 standard in a distributed manner. We show that distributed control can meet the requirements of real-time services. Simulations have proved that with proper admission control the proposed extension satisfies the requirement for low delay of real-time traffic.

The elastic traffic extension modifies the CW computation of the DCF mode of the standard. This modification has been done in such a way that the proposed architecture is backwards compatible, i.e. terminals conforming to the current standard are supported. The simulations performed have shown that the extension proposed for elastic traffic achieves the desired level of differentiation between elastic flows in a wide variety of scenarios.

Chapter 8

Implementation

In this chapter we present an implementation of the User Fair Differentiation architecture for a PC-based router running under the Linux operating system. We describe the design and implementation issues of the different components of the architecture, and validate them by measurements. The evaluation of these measurements shows that the resource allocation obtained with the proposed architecture is the desired.

The Linux operating system is a good basis for implementing a router with the UFD functionality. It runs on standard PC hardware and source code of the kernel is widely available. In addition, it supports a variety of queueing disciplines for output queues of network devices, and all functions for routing are already provided.

To get a deeper inside into our concrete realization of the UFD architecture, understanding how Linux handles basic network functions is important. Hence, a short overview of the standard Linux implementation is given next. Afterwards, the implementation of the UFD architecture is presented.

8.1 Linux network implementation

To give a short overview of the Linux IP network implementation, the course of an IP-packet through the system is described first (see Figure 8.1).

After a packet is received by a network interface card, a hardware interrupt is triggered. As a consequence, an interrupt handling routine (named `ei_interrupt()`) is invoked and determines the type of interrupt. For interrupts caused by incoming packets a further handling routine is called (`ei_receive()`) which simply copies the packet from the network card into an internal socket buffer structure and calls a procedure named `netif_rx()`. The latter queues the packet (represented by a socket buffer structure) into

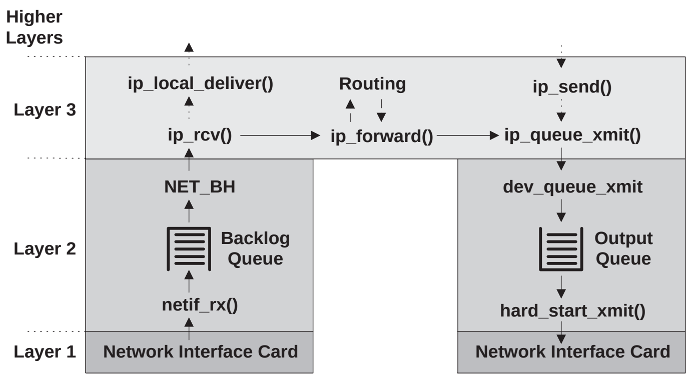

Figure 8.1: The course of a packet through the system.

a central queue (backlog queue) consisting of all packets that arrived on any network adapter of the system. The first time-critical part of the interrupt routine, called 'top-half' is finished at this time.

The necessary second part, called 'bottom-half', is handled by the network bottom-half routine (`NET_BH`) which is regularly invoked by the kernel scheduler. At first, this procedure checks whether there are any packets waiting for transmission in any output queue of any network adapter. If there are any packets waiting they are processed for a limited period. Subsequently, `NET_BH` proceeds with the next packet of the backlog queue and determines the appropriate protocol to handle the packet which is in our case the Internet Protocol IP. `ip_rcv()` checks for correctness of the IP header and then processes any existing options. It also reassembles the original IP packet from fragments if necessary and if the packet has reached its final destination. In the latter case, the packet is delivered locally, otherwise it is routed and forwarded towards its destination. `ip_forward()` tries to find the right network adapter this packet is forwarded to next by use of a routing table. If there is a valid entry in the routing table `ip_queue_xmit()` is subsequently invoked, performing some final operations such as decrementing time-to-live values and recalculating IP header checksums. `dev_queue_xmit()` queues the packet into the output queue of the corresponding network device. At this point a special queueing discipline can be invoked. Thus, each queueing discipline constitutes one output queue for a device that is not necessarily served in a FIFO with Drop-Tail basis. Within a queueing discipline, transfer of a packet onto network media is initiated by calling `hard_start_xmit()`, which instructs the network device to send the packet.

The Linux kernel already contains various queueing disciplines apart from the standard FIFO, like Class Based Queueing, Weighted Fair Queueing or Random Early Detection to implement different network features like traffic

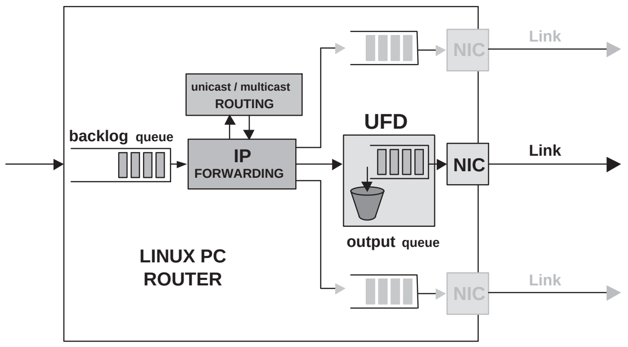

Figure 8.2: UFD implementation in Linux.

control or differentiated services (see e.g. [88, 89]).

8.2 UFD implementation

In our implementation of the UFD architecture, we inserted the algorithm of Figure 4.1 in the queuing discipline of the output queue. This algorithm decides whether an incoming packet to the output queue is enqued or dropped. This is illustrated in Figure 8.2.

The UFD queueing discipline was implemented in a kernel module. Kernel modules need not to be present all the time in the kernel, so the kernel can run without them if they are not actually used. Particularly, instead of recompiling the whole kernel and restarting the system every time a part of the module's code was changed, one can simply reload the newly coded module. This shortens development time drastically.

During the implementation of the UFD queueing discipline, a number of issues had to be solved. In the following we provide a detailed description of the various implementation issues we faced.

8.2.1 Router Performance

I/O performance and CPU router performance are crucial for successful operation, because if the PC is too slow, the protocol processing for a packet is not finished before a new packet arrives. As a consequence, the implemented UFD algorithm for packet dropping is never used because packets are dropped already earlier in the backlog queue. Nevertheless, we checked that a PC with a AMD-K6 CPU running at 350 MHz (the machine we used as a router) is sufficient to route incoming traffic of 20 Mbps at least.

8.2.2 Router Configuration

One of the parameters of the UFD algorithm that needs to be configured is the capacity of the link C. In order to set this parameter, we measured the net capacity obtained in the 10 Mbps Ethernet link with a FIFO queue. The packet lengths considered to compute this capacity included the 42 bytes of overhead of the UDP, IP and Ethernet headers and the 4 bytes of the Ethernet checksum, in addition to the packet payload. Considering this overhead, the net capacity measured was of 9.8 Mbps; this is the value we used for C in the UFD algorithm.

8.2.3 Label Location in Packet Header

An important issue of the implementation is how to insert the label value L_k into the packet header. Two possibilities are: (1) introduce a new IP option, or (2) introduce a new header between layer 2 and layer 3, similar to the way labels are transported in Multi-Protocol Label Switching (MPLS). While both of these solutions are quite general and can potentially provide large space for encoding the label, for the purpose of our implementation we considered a third option: store the state in the IP header. By doing this, we avoid the penalty imposed by most IPv4 routers in processing the IP options, or the need of devising different solutions for different technologies as it would have been required by introducing a new header between layer 2 and layer 3.

The biggest problem of using the IP header is to find enough space to insert the label. The main challenge is to remain compatible with current standards and protocols. In particular, we want to be transparent to end-to-end protocols. One possibility to achieve this goal is to use the type of service (TOS) byte. However, as we discuss in the following section, the 8 bits obtained with this option is not sufficient to encode the label with the desired level of accuracy. Another option is to use *IP identifier* field, which has 16 bits. This field is unused for non-fragmented packets (fragmented packets are identified by the pair *more fragment* and *fragment offset*). As pointed out in [90], very few packets are actually fragmented in the Internet (0.22%). In the UFD implementation, we have chosen this latter option for the labeling. Fragmented packets are ignored and forwarded as usual.

8.2.4 Label Mapping

The next issue to solve was how to map the label values (which theoretically can be any real value) into the 16 bits of the *IP identifier* field. To represent a wide range of rates in the Internet while maximizing the accuracy, we used a logarithmic scale to map the labels between L_{min} and L_{max} to discrete integer values between 0 and $2^{16}-1$. Let L be the original label (real value) and V its integer representation in the 16 bit field. Then,

$$V = \left\lfloor (2^{16}-1)\frac{log_2(L) - log_2(L_{min})}{log_2(L_{max}) - log_2(L_{min})} \right\rfloor \tag{8.1}$$

The above mapping had already been proposed in [35] but with log_{10} instead of log_2. The reason to use log_2 is because working in base 2 allows to perform operations more efficiently. Specifically, the computation of 2^x, which is required at core nodes to unmap label values, can be very easily performed by shifting x positions the bit representation of 1.

With Equation 8.1, we can map labels between 1 (L_{min}) and 2^{32} (L_{max}) with an error bounded by 0.04%. Note that, using the 8 bits of the TOS field instead of the 16 bits of the *IP identifier* field, this error bound would be of 9.09%, which is unacceptably high.

8.2.5 Rate Estimation at Core Nodes

The last issue we had to solve for the implementation was the rate estimation. For the estimation of A and F at core nodes we decided to use the *Time Sliding Window* (TSW) algorithm [91] (Equation 8.2) instead of the exponential averaging of Equation 3.2. The reason why we chose to use the TSW algorithm was to avoid expensive exponential computations at core nodes. The experimental results show that this change does not affect the accuracy of the estimation of L_{fair}.

$$r_{new} = \frac{l_k}{T_k + K} + \frac{K}{T_k + K} \cdot r_{old} \tag{8.2}$$

8.3 Experimental Results

To validate the Linux implementation and evaluate the performance of the UFD architecture with a real implementation we performed some tests. For these tests we used the testbed already explained in Chapter 6 (see Figure 6.5). Tests were run for two types of traffic: CBR traffic and ON/OFF. Packets had a constant UDP payload length of 1000 bytes.

	TEST 1		TEST 2		TEST 3	
user	share	S (Kbps)	share	S (Kbps)	share	S (Kbps)
user A	1	4904	2	6513	3	7315
user B	1	4895	1	3278	1	2472

Table 8.1: Bandwidth Allocation with CBR traffic.

8.3.1 CBR traffic

In order to validate the bandwidth allocation resulting from our implementation, we first ran some tests with CBR traffic. In these tests, both senders A and B consisted of one user (user A and user B) sending each a CBR flow at a rate of 10 Mbps.

Table 8.1 shows the bandwidth allocation resulting from changing the share assigned to user A in the above scenario. The results obtained validate the implementation, since bandwidth is distributed among users A and B proportionally to their shares.

8.3.2 ON/OFF traffic

Bandwidth allocation is much easier when dealing with CBR traffic than when dealing with bursty traffic. To show the behavior of the UFD architecture in the latter case, we performed the following experiment.

Sender A consisted of one user sending a CBR flow at a rate of 10 Mbps. Sender B consisted of one user sending an ON/OFF flow with periods ON and OFF exponentially distributed (average $T_{ON} = T_{OFF} = T$). The sending rate in the ON period was 10 Mbps. Both users had a share of 1.

Figure 8.3 shows the results of the above tests (throughput obtained by sender B). For values of T smaller than K (100 ms), the ON/OFF flow obtains its almost full share of bandwidth (i.e. as if it was sending CBR traffic). However, for larger values of T, the flow's throughput tends asymptotically to a half of the fair bandwidth share.

From the above experiment we conclude that in the UFD architecture, the averaging constant K expresses the order of magnitude of the accepted level of burstiness in a user[1].

[1]Note that, in contrast to the results obtained in this section, in the simulation results presented in Section 3.6.8, bursty users were not penalized for their bursty behavior. This is because the traffic of those users consisted of an aggregation of ON/OFF sources, which results in a much less bursty behavior than the single ON/OFF source of this section.

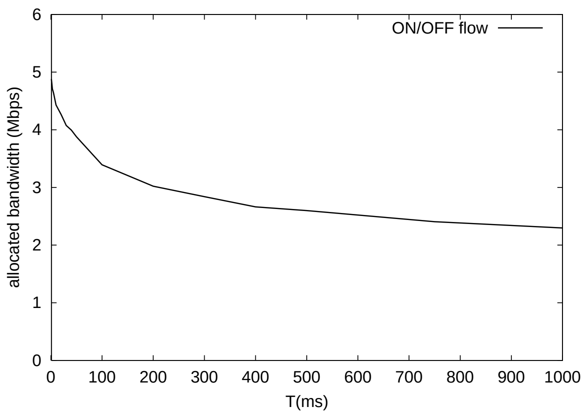

Figure 8.3: Bandwidth Allocation with ON/OFF traffic.

Chapter 9

Conclusions

Over the last decade, a considerable effort has been invested to augment the functionality of the current Best Effort Internet. For this purpose, the IntServ and DiffServ architectures have been proposed. These architectures provide new functionality by changing the packet switched paradigm of IP (IntServ emulates circuit switched networks while DiffServ relies on route pinning combined with admission control). However, past experiences have shown difficulties in changing the packet switched paradigm of the Internet; nowadays, the deployment of IntServ is almost unanimously discarded, while the deployment of DiffServ is highly questionable.

In this thesis we have proposed a novel architecture to augment the functionality of the Best Effort model that, as opposed to IntServ and DiffServ, does not change the paradigm of the current Internet. Keeping the packet switched paradigm strongly contributes to the simplicity of the architecture, as reflected by the following features:

No Signaling The proposed architecture requires no signaling between the user and the network. The contract between the users and the network is based on the shares, which are configured statically at the network's ingress.

Routing Independence The architecture does not interact with the routing; instead, it takes the routing as a given input.

No accounting Since the price paid by the users is based merely on the shares contracted and is usage independent, there is no need for accounting.

Scalability Since no per-flow or per-user state is required at core nodes, the architecture scales well with the number of users.

Despite its simplicity, the proposed architecture still adds major functionality to the current Internet architecture:

User fairness Resource allocation in the current Internet is based on TCP fairness, which allocates network resources on a flow basis. However, since the user is the entity to which commonly pricing schemes apply, resources should be allocated on a user basis. To accomplish this, the proposed architecture allocates resources in a *user fair* way.

Service differentiation In today's Internet there is a growing demand for service differentiation. For example, there are companies relying on the Internet for the day-to-day management of their global enterprise. These companies are willing to pay a substantially higher price for the best possible service level from the Internet. At the same time, there are millions of users who want to pay as little as possible for more elementary services. In the proposed architecture, the contracted share serves as the basis for service differentiation.

Comprehensible charging The price paid by a user depends on the service level contracted, expressed by the share, and is fixed. A major advantage of this flat rate pricing is its comprehensibility.

Real-time traffic The Internet was originally designed for elastic traffic and is not well adapted to support real-time traffic. In contrast, our architecture has been designed to satisfy the requirements of this traffic type.

Unfriendliness proof The current Internet model relies on the applications' friendly behavior to fairly share the network resources among the users. Therefore the cooperation of the end systems (such as provided by TCP congestion control mechanisms) is vital to make the system work. In today's Internet, however, such dependence on the end systems' cooperation for resource distribution is increasingly becoming unrealistic. Given the current best-effort model with FIFO queueing inside, it is relatively easy for non-adaptive UDP sources to gain greater shares of network bandwidth and thereby starve other, well-behaved, TCP sources. In contrast, this is not possible in our architecture.

Multicast Incentive Even though multicast strongly contributes to save bandwidth, it is rarely used. Among the reasons that slow down the use of multicast we find the lack of incentive for it. This problem is corrected in the proposed architecture.

We believe that the combination of *simplicity* —compared to IntServ and DiffServ— and *functionality* —compared to the current Internet— of the proposed architecture makes it a suitable candidate for the next step in the evolution of the Internet.

The performance of the architecture and the different extensions (real-time, multicast and wireless) have been validated via:

- Simulations with the ns-2 tool.

- Experiments with a Linux implementation.

Both simulation and experimental results have shown that the distribution of network resources achieved with the proposed architecture is fairly close to the theoretical. We conclude that a real implementation of the proposed architecture is feasible, can be done efficiently and provides good results.

Acknowledgements

I gratefully acknowledge the support and collaboration of a number of people without whom this PhD thesis would not have been possible:

Professor Sebastià Sallent, my PhD advisor, gave me the opportunity of studying my PhD remotely from Germany. We have had many interesting and fruitful discussions about the technical contents of this thesis, and we coauthor a number of papers (4). His guidance has been of key importance for the completion of my thesis.

My boss at NEC, Dr. Heinrich J. Stuettgen, offered me the possibility of combining my job at NEC with my PhD studies, and has been very supportive throughout the whole duration of my thesis. The synergy between my projects at NEC and the contents of this thesis has been extremely helpful.

I had a lot of interesting discussions about fairness issues in computer networks with Robert Denda, PhD student at University of Mannheim. Robert is coauthor of two of the papers on which this thesis is based.

Dr. Christoph Kuhmuench provided the layered video software that has been used in this thesis. He is coauthor of a paper that studies packet dropping policies for layered video.

The design of NEC's QoS extensions for Wireless LAN was done together with Markus Radimirsch from University of Hannover. Markus focused on the design of the real-time part of the wireless architecture.

Dr. Sandra Tartarelli worked together with me in NEC's DiffServ simulation project. She has read an earlier draft of this thesis and provided helpful comments.

A number of students from the Universitat Politècnica de Catalunya have contributed to the results of this PhD with their Master theses. Olga Leon studied via simulation the performance of the proposed architecture. Frederic Raspall worked on packet dropping policies for layered video. Joaquim Esteve investigated different ways of providing QoS in Wireless LAN. Xavier Pérez evaluated the wireless part of the architecture via simulation. David Anguera worked on implementation issues. I coauthor papers with Olga, Frederic, Xavier and David.

I would like to thank all the people mentioned above for their help and collaboration. It has been a pleasure for me to share my work with them.

Bibliography

[1] D. Bertsekas and R. Gallager. *Data Networks*, chapter 6, pages 524–529. Prentice-Hall, 1987.

[2] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8(1):33–37, January 1997.

[3] R. Denda, A. Banchs, and W. Effelsberg. The Fairness Challenge in Computer Networks. In *Proceedings of the 1st International Workshop on Quality of future Internet Services (QofIS 2000)*, Berlin, Germany, September 2000.

[4] Scott Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal Selected Areas Communication*, 13(7):1176–1188, September 1995.

[5] A. Banchs. User Fair Queuing: Fair Bandwidth Allocation for Users. Accepted to IEEE INFOCOM 2002.

[6] A. Banchs, O. Leon, and S. Sallent. The Olympic Service Model: Issues and Architecture. In *Proceedings of the 2nd International Workshop on Quality of future Internet Services (QofIS 2001)*, Coimbra, Portugal, September 2001.

[7] O. Leon. Simulation Study of the Scalable Share Differentiation Architecture. Projecte Final de Carrera, Universitat Politècnica de Catalunya, September 2001. Supervised by S. Sallent and A. Banchs.

[8] A. Banchs and R. Denda. A Scalable Share Differentiation Architecture for Elastic and Real-Time Traffic. In *Proceedings of the Eight IEEE/IFIP International Workshop on Quality of Service (IWQoS 2000)*, Pittsburg, PA, June 2000.

[9] A. Banchs, F. Raspall, D. Anguera, and S. Sallent. Fair Bandwidth Allocation for Multicast and Unicast Flows. Submitted.

[10] F. Raspall, C. Kuhmuench, A. Banchs, F. Pelizza, and S. Sallent. Study of packet dropping policies on layered video. In *Proceedings of Packet Video Workhsop*, Korea, April 2001.

[11] F. Raspall. Impact of Packet-dropping Policies into Video Quality in Layered Transmissions. Projecte Final de Carrera, Universitat Politècnica de Catalunya, February 2001. Supervised by S. Sallent and A. Banchs.

[12] A. Banchs, X. Pérez, M. Radimirsch, and S. Sallent. Service Differentiation Extensions for IEEE 802.11. In *11th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2001)*, Boulder, CO, March 2001.

[13] A. Banchs, X. Pérez, M. Radimirsch, and H. Stuettgen. Service Differentiation Extensions for Elastic and Real-Time Traffic in 802.11 Wireless LAN. In *Proceedings of the IEEE Conference on High Performance Switching and Routing (HPSR 2001)*, Dallas, Texas, May 2001.

[14] A. Banchs and X. Pérez. Distributed Weighted Fair Queuing in 802.11 Wireless LAN. Accepted to IEEE ICC 2002.

[15] X. Pérez. IEEE 802.11 Multimedia Extensions. Projecte Final de Carrera, Universitat Politècnica de Catalunya, November 2000. Supervised by S. Sallent and A. Banchs.

[16] D. Anguera. Implementation of a Core Stateless Architecture for Bandwidth Allocation. Projecte Final de Carrera, Universitat Politècnica de Catalunya, to be published. Supervised by X. Hesselbach and A. Banchs.

[17] A. Banchs and X. Pérez. Providing Throughput Guarantees in 802.11 Wireless LAN. Accepted to IEEE WCNC 2002.

[18] A. Banchs, X. Pérez, W. Pokorski, and M. Radimirsch. A Proposal for Wireless MAC Multimedia Extensions. IEEE 802.11-00/205, July 2000.

[19] A. Banchs, W. Pokorski, and M. Radimirsch. Considerations about Wireless MAC Multimedia Extensions. IEEE 802.11-00/100, May 2000.

[20] S. Sato, K. Kobayashi, H. Pan, S. Tartarelli, and A. Banchs. Configuration Rule and Performance Evaluation of Diffserv Parameters. In *Proceedings of the Seventeenth International Teletraffic Congress (ITC17)*, Salvador da Bahia, Brazil, December 2001.

[21] M. Brunner, A. Banchs, S. Tartarelli, and H. Pan. A one-to-any Probabilistic Assured Rate Per-Domain Behavior for Differentiated Services. Internet draft, April 2001.

[22] S. Tartarelli and A. Banchs. Performance Evaluation for Diffserv Parameters Configuration. Technical report, NEC, March 2001.

[23] J. Esteve. QoS Extensions to IEEE 802.11. Projecte Final de Carrera, Universitat Politècnica de Catalunya, April 2000. Supervised by S. Sallent and A. Banchs.

[24] S. Tartarelli and A. Banchs. Random Early Marking: Improving TCP Performance in DiffServ Assured Forwarding. Patent application to the German Office. Also, accepted to IEEE ICC 2002.

[25] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM'88*, pages 314–329, Stanford, CA, August 1988.

[26] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queuing Algorithm. *Internetworking Research and Experience*, pages 3–26, October 1990.

[27] S. Floyd and V. Jacobson. Link Sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[28] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, June 1994.

[29] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998.

[30] H. R. Varian. *Intermediate Microeconomics - A Modern Approach.* W. W. North & Company, New York/London, fifth edition, 1999.

[31] H. R. Varian. Distributive Justice, Welfare Economics, and the Theory of Fairness. *Philosophy & Public Affairs*, 4(3):223–247, 1975.

[32] L. Massoulie and J. Roberts. Bandwidth Sharing: Objectives and Algorithms. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.

[33] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM'98*, pages 118–130, Vancouver, Canada, August 1998.

[34] Z. Cao, Z. Wang, and E. Zegura. Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State. In *Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.

[35] A. Clerget and W. Dabbous. *T*UF: Tag-based Unified Fairness. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

[36] H. Zhu, A. Sang, and S. Li. Weighted Fair Bandwidth Sharing Using SCALE Technique. *Computer Communications Journal, Special Issue in QoS*, 24(1), January 2001.

[37] M. Nabeshima, T. Shimizu, and I. Yamasaki. Fair Queuing with In/Out Bit in Core Stateless Networks. In *Proceedings of the Eight IEEE/IFIP International Workshop on Quality of Service (IWQoS 2000)*, Pittsburg, PA, June 2000.

[38] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. Technical report, Carnegie Mellon University, June 1998. Available at `http://www.cs.cmu.edu/~istoica/csfq-tr.gz`.

[39] Z. Wang. A Case for Proportional Fair Sharing. In *Proceedings of the Sixth IEEE/IFIP International Workshop on Quality of Service (IWQoS'98)*, Napa, CA, May 1998.

[40] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(1):397–413, August 1993.

[41] UCB/LBNL/VINT. Network Simulator (ns), version 2. `http://www.isi.edu/nsnam/ns/`.

[42] R. Kapoor, C. Cassetti, and M. Gerla. Core-Stateless Fair Bandwidth Allocation for TCP flows. In *Proceedings of IEEE ICC 2001*, Helsinki, Finland, June 2001.

[43] QBone Bandwidth Broker Advisory Council home page. `http://www.merit.edu/working-groups/i2-qbone-bb`.

[44] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, June 1999.

[45] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. RFC 2598, June 1999.

[46] C. Kalmanek, D. Shur, S. Sibal, C. Sreenan, and J. Merwe. NED: A Network-Enabled Digital Video Recorder. In *11th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2001)*, Boulder, CO, March 2001.

[47] K. Kilkki. Simple Integrated Media Access. draft-kalevi-simple-media-access-01.txt, Internet draft, June 1997.

[48] M. Loukola, J. Ruutu, and K. Kilkki. Dynamic RT/NRT PHB Group. draft-loukola-dynamic-00.txt, Internet draft, November 1998.

[49] CISCO SYSTEMS. Congestion Avoidance Overview. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr%/fqos\_c/fqcprt3/qcfconav.htm`.

[50] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. In *Proceedings of ACM SIGCOMM'99*, pages 109–120, Cambridge, MA, September 1999.

[51] T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. Relative Delay Differentiation and Delay Class Adaptation in Core-Stateless Networks. In *Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.

[52] Y. Moret and S. Fdida. A Proportional Queue Control Mechanism to Provide Differentiated Services. In *International Symposium on Computer System*, Belek, Turkey, October 1998.

[53] WTP packet scheduler for ns2. `http://www.cis.udel.edu/~dovrolis/ns-WTP.tar`.

[54] A. Feldmann, A. C. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of the Internet WAN traffic. In *Proceedings of ACM SIGCOMM'98*, pages 25–38, Vancouver, Canada, August 1998.

[55] Diffserv Model for the ns2 simulator. `http://www7.nortel.com:8080/CTL/`.

[56] K. Nichols, V. Jacobson, and L. Zhang. A Two-bit Differentiated Services Architecture for the Internet. RFC 2638, July 1999.

[57] J. Y. Le Boudec P. Hurley. A proposal for an asymmetric best-effort service. In *Proceedings of the Seventh IEEE/IFIP International Workshop on Quality of Service (IWQoS'99)*, pages 132–134, London, England, May 1999.

[58] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource Reservation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, September 1997.

[59] L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, and H. Zhang. Endpoint Admission Control: Architectural Issues and Performance. In *Proceedings of ACM SIGCOMM 2000*, pages 57–70, Stockholm, Sweden, August 2000.

[60] G. Bianchi, A. Capone, and C. Petrioli. Throughput analysis of end-to-end measurement-based admission control in IP. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[61] V. Elek, G. Karlsson, and R. Ronngren. Admission control based on end-to-end measurements. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[62] F. Kelly, P. Key, and S. Zachary. Distributed admission control. *IEEE Journal on Selected Areas in Communications*, 18(12), December 2000.

[63] S. E. Deering. Multicast routing in internetworks and extended LANs. In *Proceeding of ACM SIGCOMM'88*, pages 55–64, Stanford, CA, August 1988.

[64] H. Eriksson. MBONE: The Multicast Backbone. *Communications of the ACM*, 37(8), August 1994.

[65] A. Legout, J. Nonnenmacher, and E. Biersack. Bandwidth Allocation Policies for Unicast and Multicast Flows. *IEEE/ACM Transactions on Networking*, 9(4), August 2001.

[66] H. W. Holbrook and D. R. Cheriton. IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications. In *Proceedings of ACM SIGCOMM'99*, pages 65–78, Harvard, MA, September 1999.

[67] C. Kumuench, G. Kuehne, C. Shremmer, and T. Haenselmann. A video-scaling algorithm based on human perception for spatio-temporal stimuli. In *Proceedings of SPIE, Multimedia Computing and Networking*, 2001.

[68] T. Kim, R. Sivakumar, K.-W. Lee, and V. Bharghavan. Multicast Service Differentiation in Core-Stateless Networks. In *Proceedings of International Workshop on Networked Group Communication*, Pisa, Italy, November 1999.

[69] S. McCanne and V. Jacobson. Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM'96*, Stanford, CA, August 1996.

[70] A. Flores and M. Ghanbari. Prioritised delivery of layered coded video over IP networks. *ACM Transactions on Multimedia*, 2001.

[71] D. Rubenstein, J. Kurose, and D. Towsley. The Impact of Multicast Layering on Network Fairness. In *Proceedings of ACM SIGCOMM'99*, Boston, MA, September 1999.

[72] IEEE. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Standard 802.11, June 1999.

[73] F.A. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part I - Carrier Sense Multiple-Access Modes and their throughput delay characteristics. *IEEE Transactions on Communications*, 23(12):1417–1433, 1975.

[74] F.A. Tobagi and L. Kleinrock. Packet switching in radio channels: Part II - the Hidden Terminal Problem in Carrier Sense Multiple-Access Modes and the Busy-Tone Solution. *IEEE Transactions on Communications*, 23(12):1417–1433, 1975.

[75] C. Fullmer and J. J. Garcia-Luna-Aceves. Floor Acquisition Multiple Access (FAMA) for Packet Radio Networks. In *Proceedings of ACM SIGCOMM'95*, Cambridge, MA, August 1995.

[76] T. Nandagopal, S. Lu, and V. Bharghavan. A Unified Architecture for the Design and Evaluation of Wireless Fair Queuing Algorithms. In *Proceedings of ACM MOBICOM'99*, Seattle, WA, August 1999.

[77] S. Lu, V. Bharghavan, and R. Srikant. Fair Scheduling in Wireless Packet Networks. In *Proceedings of ACM SIGCOMM'97*, Cannes, France, August 1997.

[78] M. Barry, A. Veres, and A. T. Campbell. Distributed Control Algorithms for Service Differentiation in Wireless Packet Networks. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

[79] A. Ayyagari, Y. Bernet, and T. Moore. IEEE 802.11 Quality of Service - White Paper. IEEE 802.11-00/028.

[80] A. Imad and C. Castelluccia. Differentiation Mechanisms for IEEE 802.11. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

[81] N. H. Vaidya, P. Bahl, and S. Gupta. Distributed Fair Scheduling in Wireless LAN. In *Proceeding of ACM MOBICOM 2000*, Boston, MA, August 2000.

[82] V. Kanodia, C. Li, B. Sadeghi, A. Sabharwal, and E. Knightly. Distributed Multi-Hop with Delay and Throughput Constraints. In *Proceedings of ACM MOBICOM 2001*, Rome,Italy, July 2001.

[83] J.L Sobrinho and A.S. Krishnakumar. Real-Time Traffic over the IEEE 802.11 Medium Access Control Layer. *Bell Labs Technical Journal*, Autumn 1996.

[84] M. A. Visser and M. E. Zarki. Voice and Data transmission over an 802.11 Wireless network. In *Proceeding of PIMRC*, Toronto, Canada, September 1995.

[85] ETSI. Broadband radio access networks (BRAN); HIgh PERformance Radio Local Area Network (HIPERLAN) Type 1; Functional Specification. European Norm 300 652 (V1.2.1), ETSI, 1998.

[86] J. Khun-Jush, G. Malmgren, P. Schramm, and J. Torsner. HIPERLAN type 2 for broadband wireless communication. Ericsson Review, no.2 (see http://www.ericsson.com/review), 2000.

[87] S. Chevrel et al. Analysis and Optimisiation of the HIPERLAN Channel Access Contention Scheme. Wireless Personal Communications 4, pp. 27-39, Kluwer Acadamic Publishers, 1997.

[88] W. Almesberger. Traffic Control implementation overview. `ftp://lrcftp.epfl.ch/pub/people/almesber/tcio-current.ps.gz`.

[89] K. Wehrle R. Bless. Evaluation of differentiated services using an implementation under linux. In *Proceedings of the Seventh IEEE/IFIP International Workshop on Quality of Service (IWQoS'99)*, London, England, May 1999.

[90] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings of ACM SIGCOMM'99*, pages 81–94, Boston, MA, September 1999.

[91] D. D. Clark and W. Fang. Explicit Allocation of Best Effort Packet Delivery Services. *IEEE/ACM Transactions on Networking*, 6(4):362–373, August 1998.