Controlled Multimedia Wireless Link Sharing via Enhanced Class-Based Queuing with Channel-State-Dependent Packet Scheduling

Christine Fragouli
Vijay Sivaraman
Mani B. Srivastava
University of California, Los Angeles

ABSTRACT: A key problem in transporting multimedia traffic across wireless networks is a *controlled sharing* of the wireless link by different packet streams. So far this problem has been treated as that of providing support for quality of service in time division multiplexing based medium access control protocols (MAC). Adopting a different perspective to the problem, this paper describes an approach based on extending the *Class-Based Queuing* (CBQ) based controlled hierarchical link sharing model proposed for the Internet [Floyd95]. Our scheme enhances CBQ, which works well in wired links such as point-to-point wires of fixed bandwidth, to also work well with wireless links based on radio channels that are (i) inherently shared on-demand among multiple radios, and (ii) are subject to highly dynamic bandwidth variations due to spatially and temporally varying fadings with accompanying burst errors. The proposed scheme is based on combining a modified version of CBQ with *Channel-State Dependent Packet Scheduling* [Bhagwat96].

1.0 Introduction

A key problem in the emerging wireless multimedia packet networks such as [Agrawal96] is providing a mechanism for *controlled sharing* of the wireless link by different packet streams, belonging to different applications and in general traversing to or from different radios, that are sharing the link. In research literature, this controlled wireless link sharing problem has largely been treated as that of providing support for quality of service (QoS) in medium access control protocols (MAC). Various proposed QoS-aware MAC protocols [Karol95, Sivalingam97] typically accomplish this goal by providing a centralized channel bandwidth reservation and scheduling mechanism on top of a basic time division multiplexing (TDM) structure with time divided into slots and frames. While providing a QoS framework, such MAC protocols also have complexities and inefficiencies associated with their underlying virtual connection-oriented rigid temporal structure. While such an approach might be good for overlaying multimedia on top of the cellular phone network or for the virtual connection oriented wireless ATM networks, it does not fit well with mobile IP based wireless LANs using radios such as WaveLAN and RangeLAN. Such radios, which are usually based on the 802.11 MAC protocol or similar CSMA/CA variant MAC protocols, are being used in laptops for wireless LAN access to the Internet.

1.1 Our approach

We have therefore taken a different approach to the problem, viewing it not as a QoS-based MAC problem but as a problem of making link sharing in Internet work in a wireless radio link. Specifically, we have developed a scheme based on extending the *Class-Based Queuing* (CBQ) based controlled hierarchical link sharing model that has been proposed by Floyd and Jacobson [Floyd95]. CBQ is intended to enable internet routers to control distribution of bandwidth on local links in response to local needs while retaining the decentralized flavor of the Internet. Controlled link sharing via CBQ, by allowing different traffic types to be isolated, is envisioned to work with priority-based packet scheduling to meet end-to-end real-time service requirements on the Internet.

Our scheme enhances CBQ, which works well in wired links, such as point-to-point wires of fixed bandwidth, to also work well with wireless links based on radio channels that are (i) inherently shared on-demand among multiple transceivers, and (ii) are subject to highly dynamic bandwidth variations due to spatially and temporally varying fadings with accompanying burst errors. Our proposed controlled wireless link sharing scheme is based on combining a modified version of CBQ with *Channel-State Dependent Packet Scheduling* (CSDPS) [Bhagwat96]. Intuitively, the CBQ component of our scheme provides the controlled sharing among multiple packet streams (fairness) while the CSDPS component improves channel utilization (throughput) by taking into account the different states of the wireless link radio channel that may be seen by the different spatially distributed receivers. This allows our scheme to simultaneously achieve controlled sharing of the wireless link and improved radio channel utilization.

A wireless link sharing scheme has to satisfy the twin goals of (i) fairness, or controlled sharing of bandwidth among multiple packet streams, and (ii) throughput, or high utilization of available radio channel bandwidth. However, in order to achieve these goals a link-layer protocol for a wireless environment has to deal with problems associated with the radio propagation characteristics:

- A transmitter cannot know by its own means the interference, fading and general reception quality at the receiver. A variety of link-layer protocols have been proposed [Karn90] for this problem, the most common solution being the exchange of RTS and CTS (*request to send* and *clear to send*) control messages, which we have also adopted.
- If a transmitter communicates with N different terminals, then it actually has to deal with N spatially distinct independent per-destination links over a shared wireless radio channel. A "link" here refers to the wireless path between a specific pair of transmitter and receiver. The error behavior of the link between the transmitter and each receiver is bursty, and both time and space dependent. For example, only a subset of the receivers may be in a fade zone or be subjected to interference at any given time.

When a transmitter has a FIFO queue of packets to transmit to different receivers, through different links, a Head Of Line (HOL) problem may occur [Bhagwat96]. Repeated RTS-CTS attempts are done by MAC protocols with link level retransmissions such as 802.11 and CSMA/CA. The transmission of the head of line packet to a specific receiver may repeatedly fail if the link to that receiver is in a burst error state. This can happen if the specific receiver is in a fade, or if there is another transmitter causing interference at that receiver (e.g., due to frequency collision in slow frequency hopping ISM band radios). This results in a blocking of the transmission of packets that are further down the queue but destined to other receivers. Thus, all receivers suffer even though only the link to the receiver for the packet at the head of the FIFO queue is in a bad condition. Since the wireless links to various destinations are statistically independent, packets for other destinations could have been successfully transmitted during this interval. Our CBQ and CSDSP based link sharing scheme provides for controlled sharing of the wireless link while coping with these intrinsic problems.

1.2 Related work

Most relevant to our scheme is the prior research on Class Based Queueing for controlled (wired) link sharing for Internet, and on Channel State Dependent Packet Scheduling for improved wireless link performance. Link-level channel-state-dependent packet scheduling (CSDPS) was proposed in [Bhagwat96] to overcome the aforementioned problem of head of line blocking and consequent poor wireless link utilization. The key-function of the state dependent scheduler, is that the terminal abandons the transmission effort for the HOL packet as soon as it is "persuaded" that the destination is for the time being not-reachable, proceeds with the transmissions in the rest of the receivers and retries sending the abandoned packet at a later time, when the link might have improved. Although it attempts to maximize channel utilization, CSDPS by itself, however, does not provide any mechanism for making or meeting bandwidth commitments to different connections or groups of connections in a wireless link. It is unfair in this sense and needs to be associated with a link scheduler to ensure fairness.

Class Based Queuing (CBQ) link sharing mechanism was proposed in [Floyd95] as the key component in enabling the deployment of priority based packet scheduling algorithms to support end-to-end service requirements of real time traffic in the Internet. CBQ associates a hierarchical structure with a link, thus aggregating the packet streams belonging to different connections into classes consisting of one or more connections. CBQ associates quantitative bandwidth commitments with the hierarchical class organization and strives to provide controlled link sharing by ensuring fairness in the sense that each interior and leaf class gets its allocated bandwidth over a relevant time interval. As a secondary goal of CBQ is to provide a guideline for distributing excess available bandwidth among the classes instead of an arbitrary allocation. A brief summary of the main functions and features of CBQ [Floyd95] relevant to our scheme is presented in Section 3.4.

2.0 Problem description

We are interested in the aforementioned architecture: one server (transmitter), serving a number of different applications (queues), each of which is destined to a possibly different receiver. For example, a base-station transmitting downstream to the different terminals in his cell, or one terminal, on which run a number of applications each communicating with a different peer, possibly to a different mobile through a different link or wireless path from the receiver. We propose leveraging CBQ and combining it with a CSDPS module, to achieve *improvement of throughput* and *fairness* over a shared wireless channel.

In this architecture, we have two resources:

- The server/transmitter time: the server is sequential (for example one radio transmitter that can transmit one packet at a time).
- The bandwidth of the radio channel.

To each queue is allocated a specific percentage of bandwidth of the shared channel. If all the transmitter-receiver links were error free, then the assigned percentage of bandwidth would correspond to the percentage of time the server devotes to each queue. CBQ could then be employed to assure that every queue takes its demanding percentage of the server's time. In reality radio channels are bursty due to fadings and frequency collisions. Their behavior can be often modeled as a Markov chain of two or more states [Bhagwat96, Swarts94, Wang95]. Transmitting unsuccessful RTS-CTS to a link in a bad state causes the server to spend time on a queue/link without improving its throughput. The CSDPS module tries to assure that if possible the server will only spend time transmitting to each link when this link is in a good state while CBQ ensures fairness.

Background: RTS-CTS handshake in wireless links

Many different variations of the RTS-CTS messages have been proposed in the literature. In this paper, we use the following approach, although any other could equally well be used: A sender, sends an RTS (ready-to-send message) to its receiver, indicating that it has data to send. The receiver may not receive the RTS correctly due to errors. If it does, it replies with a CTS (clear-to-send reply), accepting the transmission. The sender may also not receive the CTS reply, because, for example, it is currently in a bad state. In this last case too he will perceive an RTS-CTS failure. If the sender fails with the RTS-CTS transmission, it persists, up to a specific maximum number of times. The RTS-CTS message exchange has a dual role:

- *To probe the state of the receiver's link:* its duration is much smaller than that of an actual packet, and thus the throughput loss incurred in case of failure, much less. This is the primary purpose we use the RTS-CTS exchange for in our scheme.
- *To deal with the hidden/exposed terminal problem:* This is the traditional role of RTS-CTS as proposed by [Karn90].

RTS-CTS use imposes an overhead to the channel throughput. This overhead depends on the length of the transmitted packets, the channel-behavior and the specific resource allocation. Assuming, for example, that the whole RTS-CTS hand-shake consumes 20 bytes worth of the senders time, the throughput in one of our specific simulations testbed dropped as much as to 84% of the throughput we could achieve if we had perfect knowledge of the receivers link (which corresponds to the use of a zero-duration RTS-CTS).

3.0 Theoretical background

3.1 Motivation example

Let us consider the following simple case: a sender that has two packet streams to send, one to each of two different receiver. Each stream is assigned 50% of the throughput. Although they are sharing the same radio channel, the states of the links from the transmitter to each of the two receivers are independent because the two receivers may be subject to different and independent fading and interference mechanisms. For purposes of illustration in this example we will assume that the state of the links to the two receivers change deterministically between a good and bad state. Later in the paper, however, we will use more realistic Markov models for fading channels.

We declare a link to a specific receiver be in a bad state when transmission to it is very likely to fail due to a corrupted packet, and in a good state, respectively when transmission on it is likely to be successful. We assume the bad state as being completely destructive and the good error-free. Typically link layer protocols in wireless LANs employ cyclic redundancy codes (CRCs) to detect errors in a packet. Even with more powerful error correcting codes, there is a certain probability of the packet being damaged beyond repair by the code. Depending on the MAC protocol, a retransmission of the damaged packet may be attempted by the MAC protocol itself, or left to a higher protocol layer. Irrespective of the final fate of the packet, a link sharing scheme would be interested in selecting the most appropriate packet transmission time in terms of the channel state.

Consider for example having to transmit the two packet streams in a wireless link where the links to the two receivers are as shown in Figure 1. Basic CBQ (and conventional MAC protocols) is totally unaware of the states of the links to individual receivers since this is not something that happens on the wired links for which CBQ was

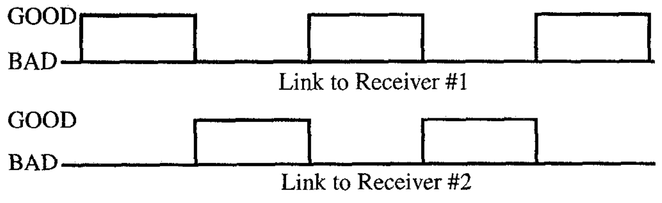

Figure 1: Links to two receivers oscillating between good and bad state

designed. In the scenario of Figure 1 CBQ would blindly assume that links to all receivers are in good state and basically treat the problem like that of sharing a fixed bandwidth wire. So, it will allow packets from each of the two streams to be blindly transmitted.

Intuitively, we would like to transmit to receiver #1 when the link to it is in a good state and link to receiver #2 is in a bad state. Similarly, when link to receiver #1 enters a bad state and link to receiver #2 the good state respectively, we would like to transmit only to receiver #2. At the same time, however, we have to regularly probe the link to the receiver to which we do not currently transmit. This probing has to be done in order to keep an updated knowledge of the states of the links to each of the receivers and to be aware of a bad to good state transition in the link to any of the receiver. This can be achieved by using channel-state-dependent (CSDPS) scheduling which has been initially introduced in [Bhagwat96]. The following subsection describes the idea behind CSDPS, and our variation.

3.2 Channel State Dependent Packet Scheduler (CSDPS)

Many studies in literature such as [Swarts94, Wang95] have established that finite state Markovian models can be effectively used to characterize the error behavior of wireless links. Based on that, the authors in [Bhagwat96] use a two-state Markov model (Good-Bad) for the link, to illustrate how using CSDPS scheduling can improve the throughput performance. Through simulation, they verify that head of line blocking causes poor utilization as well as Round Trip Time (RTT) growth and fairness problem.As a solution they propose that scheduling policies should take the wireless channel state information into consideration.These policies upon encountering a "bad link", which might be detected by the loss of a packet for example, defer transmission to that destination until the start of the next good period for the wireless link corresponding to that destination.

We adopt a variation of this approach in our work: we model the actual links as two-state Markov models. As we mentioned before, we assume that if a link is in a bad state, transmission to it is useless. Ideally we would like the mobile to have exact knowledge of the state each link is in. But this in reality never happens. So in our simulations the mobile itself keeps its estimate of how good it thinks the link is. This goodness of the link is expressed through parameter g. The mobile updates this parameter, for each link, whenever it tries to transmit packets for an application using that link. The parameter g is allowed to take a number of integer values.

The intuition is, that the better we think the link to a receiver is (the bigger the g), the more we should persist when we try to transmit to that receiver, because we indeed have better chances to succeed. On the other hand, if we think that the link to a receiver is bad, we should not lose much time on it, but instead proceed to another receiver and try this receiver again later. In our simulations g is translated to the max number of RTS attempts allowed: if a CTS is not successfully received even after g tries at sending RTS, then g is decreased exponentially. If we succeed with less than g tries, g is increased (up to a max value) inversely proportional to the number of attempts.

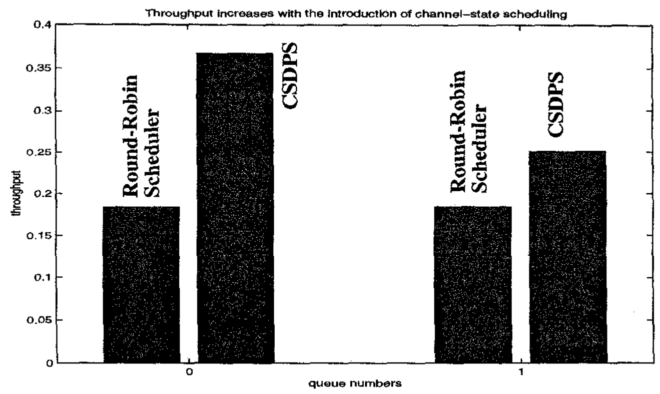

Figure 2: Throughput of Round-Robin vs. Channel State Dependent Packet Scheduler

The main discrepancy between the actual channel state and the mobile estimation stems from the fact that the mobile does not monitor each link continuously but only for a fraction of time. Then it turns its attention to other links. So when it eventually returns, the link might have completely changed behavior.

Allowing g to take more than two values increases the flexibility of the channel-state-dependent module. Multiple values of g permit the sender to be conservative and take into account its previous knowledge of the receiver's link state. In reality good state is not error-free or bad-state always destructive, so in order not to have purposeless oscillations, one would like to gradually built up the link state estimate based on more than one link probe. The performance depends on the specific functions used to update g, and is a choice for each system since no generic solution could possibly be optimum for all the different optimizations desired. Another benefit of a multi-valued g is that in our model we have assumed a two-state only Markov-model for the individual links to the various receivers sharing the wireless link. However, depending on the specific system and radio channel, a multiple state Markov model may be better. In such a case the "estimated channel state" could then be reflected in the value of g.

3.3 Need for CBQ

Referring to the motivational example used previously, consider Figure 2, where the first column represents the throughput achieved for each queue using just Round-Robin scheduling while the second column the throughput achieved using Channel-State-Dependent Packet Scheduling. The traffic used is constant bit rate (CBR). As expected from [Bhagwat96], one observes that the elimination of the HOL problem due to the use of CSDPS significantly increases the throughput for each queue. So the introduction of CSDPS is beneficial, as will indeed be verified later by our simulation results obtained under more realistic channel and traffic characteristics.

However, CSDPS with the round robin scheduler does not provide any control over the wireless link sharing. CSDPS by itself neither has any mechanism to commit specific fractions of the available bandwidth to different connections nor does it have any mechanism to enforce the allocations. It therefore is unfair in this sense. For example, it might happen that the link for a particular destination is constantly good, while the link for another destination suffers from severe fades. CSDPS simply does round robin scheduling among those receivers that currently have good connections. This can result in misbehaving applications communicating to receivers with constantly good connections (for example, if they are nearby) getting more than their fair share while

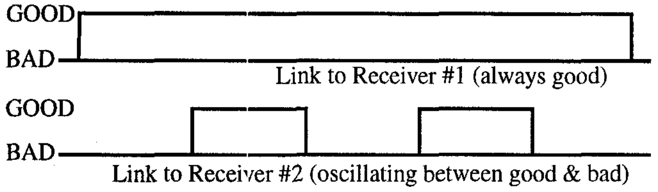
GOOD
BAD
Link to Receiver #1 (always good)
GOOD
BAD
Link to Receiver #2 (oscillating between good & bad)

Figure 3: State of links to two receivers

penalizing applications communicating to receivers whose links undergo fades and interference. A misbehaving application is one that produces more traffic than its allocated share. Furthermore, it might even happen that the link state checking period of CSDPS will always coincide with the periodic occurrence of fades.

For example, assume that the application which is communicating with receiver #1 in the previous example has a constantly good link (see Figure 3), and is misbehaving by

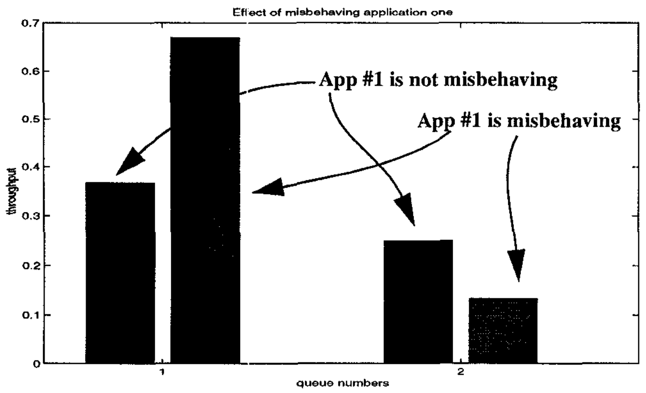

Figure 4: Sharing of throughput in CSDPS, when application #1 is not misbehaving versus when it is misbehaving

generating traffic at a rate much bigger than its allocated bandwidth share of 50%. Then, in Figure 4, we observe that it gets more than its allocated bandwidth, at the expense of the other application's throughput. The first column in Figure 4 represents the throughput when the first application is not misbehaving, and the second when it is misbehaving. The small increase in the total throughput is due to the fact that we increased the packet duration of the first application to make it misbehaving, and thus the RTS-CTS overhead decreased, resulting in an overall throughput increase, all absorbed by the misbehaving application.

The preceding example makes it clear there is a need for a mechanism in addition to CSDPS that will try to guarantee to each application that no other application will steal its allocated fair share of the bandwidth. In the context of wired networks, a mechanism that tries to ensure that different classes sharing a link receive their allocated bandwidth is provided by CBQ introduced in [Floyd95], which is presented in the following section.

3.4 Class Based Queuing

In this subsection a brief summary and terminology from [Floyd95] is presented. We assume an hierarchical link-sharing structure. All arriving packets are assigned to one of leaf classes by a *classifier*. The interior classes are used to designate guidelines about how "excess" bandwidth should be allocated, between leaf classes. The root correspond to the link which is shared by the leaf classes. A link sharing bandwidth is allocated to each class, expressed as a percentage of the overall bandwidth. The link sharing goals are the following:

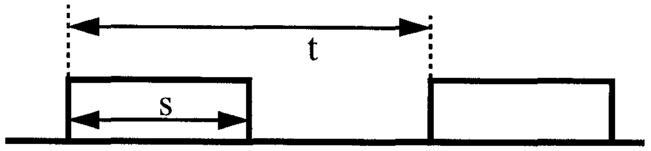

Figure 5: CBQ to estimate the throughput uses the rate of the bytes sent to calculate the inter-departure time

1. Ensure that classes receive their allocated link-sharing bandwidth over the relevant time-interval.
2. Distribute the excess of the bandwidth in a fair way.

To achieve these goals, the idea is to use a link-sharing mechanism on top of the scheduler which performs hierarchical class-based resource management (CBQ). Although conceptually we can think of it as having a link-sharing and a general scheduler, all implementations have a single integrated scheduler.

A class is called

- *overlimit* if it has recently used more than its allocated bandwidth
- *underlimit* if it has used less than a specified fraction of its allocated bandwidth
- *at limit* if it is neither of the above two
- *unsatisfied* if it is underlimit and has a persistent backlog.
- *regulated* if packets are scheduled by the link-sharing scheduler, and is restricted to transmit at no more than its allocated bandwidth, and
- *unregulated* when it is not rate-limited.

Formal link-sharing guidelines: A class can continue unregulated if one of the following conditions hold:

- The class is not overlimit
- The class has no overlimit ancestor at level *i*, and there are no unsatisfied classes at levels lower than *i*.

Otherwise, if neither of these conditions hold, i.e. the class is overlimit and some other class is unsatisfied, then the class will be regulated.

So CBQ's main functions are:

- estimate the limit status of each class (overlimit, at limit, etc.)
- determine if each queue is satisfied or unsatisfied
- if a queue is unsatisfied, restrict the appropriate other queues.

The main module used is an *estimator* which estimates the bandwidth used by each class over the appropriate time interval to determine whether or not a class has been receiving its link sharing bandwidth. An implementation of the estimator, presented in [Floyd95] which we also initially used in our simulations, is the following:

Consider a specific leaf-class/queue. Let *s* be the size of the most recently transmitted packet in bytes, *b* the link-sharing bandwidth allocated to the queue in bytes per simulation unit, and *t* the measured inter-departure time between the packet that was just transmitted and the previous packet transmitted from that queue (Figure 5) Ideally, we would like inter-departure time to be $t=s/b$. Let $diff=t-s/b$ be the discrepancy between the actual inter-departure time and the allocated inter-departure time for that class for packets of that size. So *diff* is negative when the class transmits more often than its allocated bandwidth permit, and positive if it transmits less often than allowed. The simulator computes the exponential weighted moving *avg* of the diff variable using the equation $avg \rightarrow (1-w)avg + w*diff$. A class is considered to be overlimit if *avg* is negative and underlimit if avg is positive.

The value of *avg* computed by the estimator is also used to update the *time_to_send* field associated with each queue. This field indicates the next time that the server is allowed to send a packet from that queue. For a queue with positive *avg*, the estimator sets the *time_to_send* field to zero, indicating that the class is under its limit. For a regulated class with negative *avg*, the link sharing scheduler sets the *time_to_send* field to *s/b* seconds ahead of the current time.This is the earliest time the queue will next be able to send a packet.Thus, a regulated queue is never restricted to less than its allocated bandwidth, regardless of the "excess" bandwidth used by that class in the past.

3.5 New CBQ estimator for wireless links

Using a CBQ module in our architecture, we indeed managed to suppress the misbehaving source, as can be seen in Figure 6.

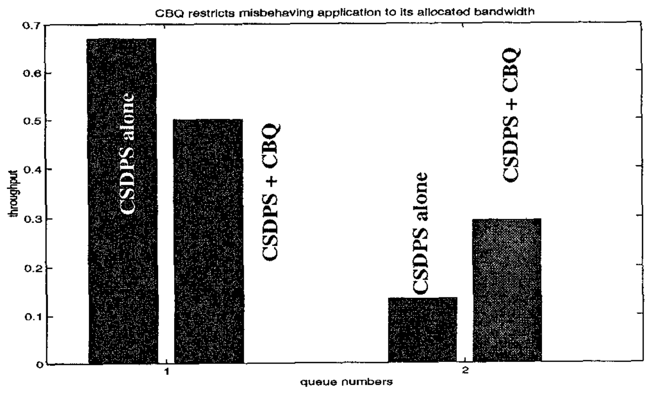

Figure 6: Throughput of CSDPS alone, versus CSDPS+CBQ

We notice though, that CBQ with an estimator implemented in the above described way (similar to [Floyd95]), provides fairness only in the sense that it restricts the misbehaving source to its allocated bandwidth of 50% of the nominal link bandwidth. However, it does not rectify the problem of a connection not getting its fair share because the link to its receiver goes into bad state.

Instead, the type of fairness that we would intuitively like to achieve in a wireless link is to guarantee, to each class queue, its allocated percentage of the *effective throughput*, where the effective throughput is defined as the total throughput achieved at each moment. That is, in the previous example, where the total throughput is around 80%, we would like each queue to be allocated exactly half of it, 40%. In other words, considering again the case in Figure 3 where link to receiver #1 is constantly good while link to receiver #2 is good only 50% of the time, we would like (i) our mobile to transmit only to receiver #1 when link to receiver #2 is in a bad state, and (ii) only to receiver #2 when the link to receiver #2 is in a good state although link to receiver #1 is in a good state too.While (ii) might appear counter-intuitive, such a strategy actually makes more sense and is fairer because it allows receiver #2 to be compensated for the bad phases underwent by the link to receiver #2.

So, we introduce the following implementation of a new CBQ estimator tailored for wireless links: A queue is defined to be unsatisfied, if it receives less than its allocated percentage of the effective throughput:

- avg=(allocated_percentage*effective_throughput)-(queue_throughput)
- effective_throughput=total_bytes_sent/time
- queue_throughput=total_bytes_sent_by_queue/time

where *time* is a specified time-frame in the past, over which we count the bytes sent. In our simulations we used the time since beginning, but an appropriate window could be chosen as well. The impact of the new CBQ estimator is illustrated by Figure 7 where the first column shows the throughputs when the original CBQ estimator from [Floyd95] is used, and the second column shows the throughput

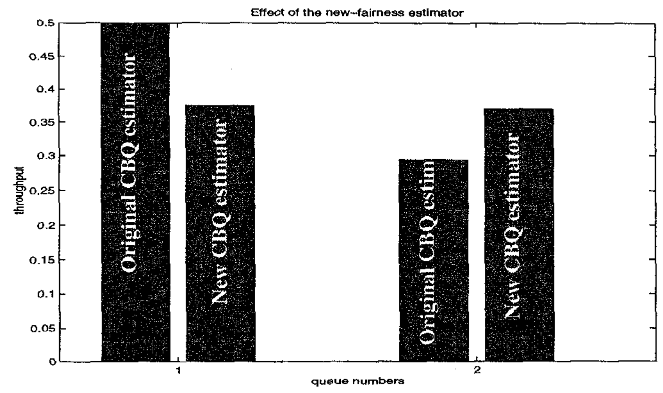

Figure 7: Throughput of original [Floyd95] vs. new CBQ estimator

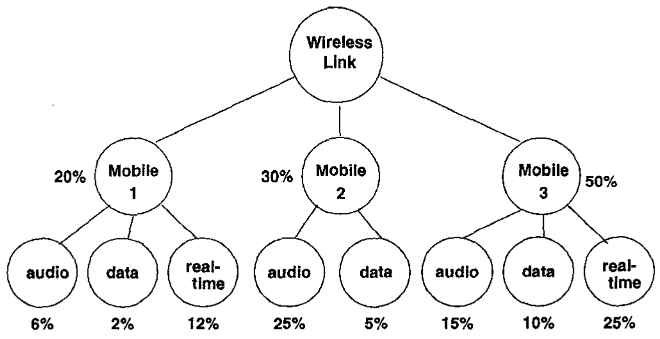

Figure 8: Throughput of Round-Robin, versus Channel-State-Dependent scheduling

when we replace the CBQ estimator by our new estimator.

The throughout is indeed now evenly split among the two applications as wished. As a consequence, a queue is not restricted up to its allocated throughput, but up to its allocated percentage of the effective throughput of the wireless channel.

Another important feature that we added to CBQ, is that we permit a restricted queue to transmit even if there exist other unsatisfied queues, provided that the restricted queue has a good link and all the unsatisfied queues have bad links. This avoids needless waste of wireless link bandwidth that results with unmodified CBQ. The simplest example to consider is when link to receiver #1 is constantly in good state and link to receiver #2 is constantly in bad state. Then, there is no reason why not to permit the application transmitting to receiver #1 to transmit as much as it wishes, even if it becomes overlimit and queue for application sending to receiver #2 is unsatisfied since link to receiver #2 is currently useless anyway. Moreover, it would be a loss of throughput and waste of resources to restrict it.

Before we discuss the reasons that further justify the need of these changes to CBQ, we present the remaining modules used in our approach so as to complete the system architecture picture.

4.0 Implementation of modified CBQ with CSDPS

The architecture we are interested in is a tree hierarchy of three levels (see Figure 8) where:

- the root stands for a transmitter (i.e., the server)
- the interior nodes stand for the link to specific spatially separated receivers to which each application wishes to transmit.
- the leaf nodes stand for specific transmitting applications.

For example, one could have a scenario where a mobile is exchanging real time speech and video with another mobile, and at the same time downloading files from a server from the wired network, or sending some data to a third mobile.

Each application's packets are put in a different queue at the transmitter. Applications generate traffic independently, although they might have the same destination. Applications that indeed send packets to the same receiver will share knowledge about the estimated state of the link to that receiver. The architecture has three main modules:

1. The channel-state determiner
2. The CBQ module
3. The scheduler

The simulations were performed using the *Maisie* discrete-event simulator[1] [Short95]. The above modules were implemented as functions that are called by the transmitting mobile module. For reasons of completeness, we repeat some of the ideas presented earlier in the first paragraph of the following subsection.

4.1 Channel-state determiner

It updates the estimation for g each time the link layer attempt to send a packet, i.e., each time an RTS-CTS exchange takes place to a specific receiver, where g corresponds to the maximum number of RTS retries allowed on the link to that receiver the next time we are going to use it. All the applications that use the same link obtain the updated value of g. If up to g tries result in failure, g is decreased exponentially, while if success occurs, g is increased inversely proportional to the number of RTS-CTS tries required for the success. The exact functions used for the g update should depend on the characteristics of each specific application.

The wireless link is modeled as a collection of independent radio link, one to each spatially separated receiver, that all temporally share the wireless link. Each independent radio link is viewed as a two-state Markov radio channel and there is one Maisie simulation module instance for each one of them. These receiver link modules change states with specific transition probabilities throughout the simulation time, with the granularity of the simulation time, independent of whether the mobile tries to transmit through them or not.

Each time an RTS is send by the mobile, the corresponding link module, depending on its state, decides if RTS is successful or not. We reiterate that CTS reception failure could also happen at the transmitter, but we assume that this probability of failure is incorporated in the link module.

4.2 The enhanced CBQ module

Our CBQ module, which is an enhancement of the model in [Floyd95], holds the following information for each packet queue:

1. *Limit status:* A queue can be *overlimit*, or *underlimit*, depending on the value of avg, where

   *avg=(allocated_percentage*effective_throughput)-(queue_throughput)*

   The limit status of each queue is recalculated each time a packet from any queue departs, as avg is a function of the total effective throughput. So actually avg is an instantaneous value, rather than an average, we just kept the name used for the respective function in CBQ.
2. *Satisfied or unsatisfied:* A queue is considered to be *unsatisfied* (unfairly treated), if it is *underlimit* (has sent less than its percentage of the effective throughput) and has a persistent backlog. Otherwise it is considered to be *satisfied.*

   A persistent backlog is declared when the queue's exponentially weighted average length (in bytes) exceeds a specific threshold. This threshold could be different for each queue. The exponentially weighted average length is updated each time a packet enters or leaves the queue. If a queue is unsatisfied, then CBQ regulates the rest of queues that are satisfied and overlimit, that is, restricts them.
3. *Restricted or not:* A queue is *restricted* if it is overlimit, and there exists at least another unsatisfied queue (a queue might be underlimit but not unsatisfied if it doesn't have a backlog, i.e. doesn't produce traffic). A restricted queue is not allowed to transmit unless there does not exist an unrestricted queue with a good link. A queue cannot be restricted unless

   (i) it is overlimit, and

   (ii) there exist at least another unsatisfied queue.

   A queue cannot be overlimit unless another one is underlimit, and vice versa. Equivalently, a restricted queue gets unrestricted, as soon as, either it is no longer overlimit, or, there no longer exist any unsatisfied queues.

4.3 Scheduler

The scheduler could implement any kind of scheduling policy such as FIFO, LIFO, priority scheduling, etc. Since we are mainly interested in observing the effect of introducing and combining the CBQ and CSDPS modules, our aim is to obtain comparative results. Therefore, we used a simple round-robin scheduler which does nothing else but checks the queues sequentially and determines which is the next queue from which a packet should be send. If all the queues are empty, the scheduler does nothing. Otherwise, it applies the following rule. A queue is allowed to transmit if:

- it is not restricted
- it is restricted, which means its fairly treated and there do exist other queues that are unfairly treated or unsatisfied, but the links to the receivers of all the unsatisfied queues are in a bad state while the link of the receiver for the restricted queue is in a good state.

If the queue currently checked is not allowed to transmit, then the scheduler goes on to the next queue, until it finds one.

4.4 Flow of events

Following is the exact sequence of events:

1. When a packet enters a queue, the queue length changes, so it is checked whether the queue now becomes unsatisfied. If it does, all overlimit (satisfied) queues become restricted.
2. The link goodness estimation is updated on RTS-CTS exchange. If RTS-CTS is successful, then a packet is sent.
3. Each time a packet is sent: (i) the sender queue length changes, so possibly the satisfied/unsatisfied status of the queue might change, (ii) the throughput of the specific queue changes as well as the total throughput (iii) the limit status of the all the queues might change, since the rate, of their throughput recalculated at that specific moment, to the total throughput has changed. So, the limit status is recalculated for all queues, and queues are made restricted or unrestricted accordingly.

5.0 Discussion of the changes introduced in CBQ

The inefficiency observed in the original (unchanged) implementation of CBQ from [Floyd95], stems from the fact that it is targeted towards a wired network while we use in a wireless link in our simulations. More specifically, we observe that:

- The total throughput in a wired environment is a known constant and thus can be allocated beforehand to the different

1. available from http://may.cs.ucla.edu/

classes. So, if each queue is restricted within its initially allocated throughput, everyone will indeed get their allocated throughput. In contrast, in a wireless environment, we cannot know the total throughput one can actually achieve beforehand, because the channels to different receivers are temporally and spatially varying. Moreover, we generally cannot even count on average values of the achieved throughput or channel behavior in long terms. That's because, each class represents a single queue communicating through its own specific link, so in contrast with a wired network, as new applications are initiated and old applications are shut down, the whole hierarchical class tree is dynamically changing, and the total throughput achieved varies accordingly. So the total throughput is not something constant, but a function of time, and we can only know its current value over a small past time interval, which we call effective throughput.

As a result, we think it makes sense to allocate to each queue a percentage of the effective throughput, or in other words, a percentage of the time the server transmits successfully. Then we must have CBQ check if this percentage is indeed allocated to each queue, every time a new packet is send, and restrict accordingly. In our simulations, we used the instantaneous value of *avg* calculated since initial time, but one could perhaps have benefits by using a *weighted average* over a window instead.

- We cannot guarantee to a queue even its allocated percentage of the throughput, because it depends not only on our scheduling choices but also on the changing link state as well. If the link to a receiver enters a bad state and gets stuck there (e.g., the receiver sits still in a fading null), one cannot do anything about it. Meanwhile, there would be no purpose served by restricting applications that are communicating with receivers to whom the links are in good states. It therefore makes sense to serve such application even though they might become or already be overlimit while applications with a bad link to receiver may be underlimit. After all, the underlimit applications cannot anyway transmit due to their link to the receiver being in the bad state.

In summary, we have introduced the following changes to CBQ:

1. We changed the CBQ estimator to adapt to the current value of the effective throughput
2. We restrict a class only if there does exist some other class that is unsatisfied and has a good link to its receiver.

6.0 Simulation of a three receiver scenario

In this section we briefly present and comment on results obtained for a case of a CBQ class tree with three different applications, each of which is transmitting to a different receiver. To provide a controlled setting for studying, we arrange for the state of the wireless links to the three receivers to change in a deterministic fashion in order to achieve a specific flows of events that we want to study. More specifically, we try to investigate the effect of a different "offset" between the state of the links to the three receivers.

We assume that all three sources have requested one third of the bandwidth each, and that the third source is misbehaving. The max throughput achieved, due to RTS-CTS overhead, if all the links are constantly in the good state, is 0.942.

We present the following two cases. In each case, in Figures 9 and 10, the first column represents what one would ideally like, the second represents the throughput achieved with CBQ+CSDPS scheduling, the third with CBQ only, the fourth with CSDPS only and the fifth when only the Round Robin scheduler is used.

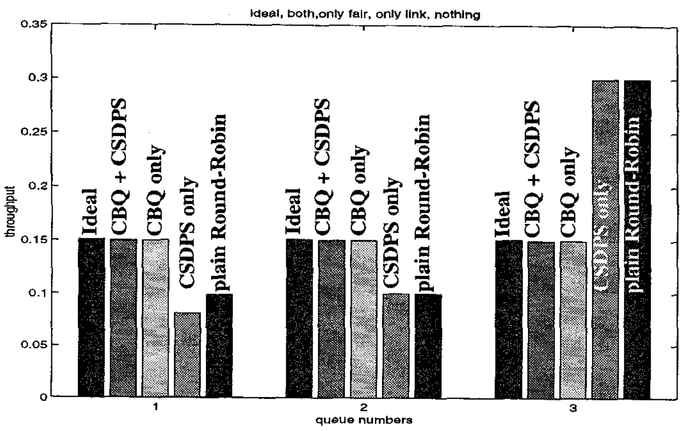

Figure 9: Throughput when all three links enter the bad and good states simultaneously

First example:

Figure 9 shows the throughput when the good-bad state transition patterns for the link to all the three receivers are deterministic and identical, with 50% duty cycle, 1000 time units in each state, and zero offset between receivers. This might be the case if a high power interferer subjects all the three receivers to interference at the same time. Intuitively, this case is identical to having a link with half throughput, because actually the links to all the receivers are good for half the time and bad for the other half. We observe that the introduction of the CSDPS scheduler does not offer any improvement, as expected, since all the links are coordinated and simultaneously enter the bad and good states. So it makes no difference whichever channel we choose to transmit. On the other hand, the introduction of CBQ does restrict the misbehaving source three to its share of the effective bandwidth. The total throughput achieved is indeed, as expected, half of the maximum possible.

Second example:

In this case, the link to each of the three receivers is successively in the good and the bad states for 1000 time units each, and the state transitions for the second and third links are phase offset from the first link by 333 and 666 time units respectively. The third source is misbehaving as before. The throughput achieved are depicted in Figure 10. We observe that, contrary to the previous case, the introduction of CSDPS scheduler does pay a lot, because there is much helpful information in the link-state. As a result the total throughput almost triples.

We also observe the effect of this special type of offset: link one, gets the least of throughput when no CBQ is used, because it has to compete for the second third of its good state period with queue two, and for the last third with both queues two and three, while queue three (corresponding to the misbehaving application) gets the most throughput, because during two thirds of its good period none else is in a good state so its the only one to get to transmit, and moreover, because it is made to misbehave by generating larger data packets, it has the least overhead due to RTS-CTS transmission. The total throughput is increased, but not to the maximum value as might be expected since the transmitter is never idle because there is much more overhead for probing unsuccessfully.

We also tried scenarios where we varied the frequency of transition between the good-bad states in the links to the various spatially distributed receivers while keeping the average time spent in each state constant. The main observation was that the more frequent the changes, the less is the total throughput because of the extra overhead associated with updating *g* more often.

7.0 Simulations results for mixed traffic scenarios

We use the structure and bandwidth allocations shown in Figure 8. A node labeled "mobile i" refers to a link for communication with the *i*-th mobile. The classes are numbered 0 through 7 starting from left to right. We used three kinds of traffic sources:

- *CBR source*, which generates packets periodically
- *Poisson source*, which generates packets with an exponentially distributed inter-arrival time
- *Bursty source*, which is characterized by two-states: an *on* state in which it generates packets with an exponential inter-arrival time, and an *off* state in which no packets are generated. The *on* and *off* states have exponentially distributed durations.

Links A and C (to mobiles 1 and 3 respectively) are fed by three sources each - one CBR, one Bursty, and one Poisson source, while link B (to mobile 2) has two sources (a CBR and a Poisson) feeding it. This is the general model that we use for simulating a mixed traffic scenario (Figure 8).

The link-sharing bandwidth of each class is indicated as a percentage of the total channel bandwidth in Figure 7. The packet sizes are fixed and the inter-packet arrival times are chosen such that each source generates traffic commensurate with its demand.

In our simulations we compare the throughput and fairness perceived by each queue under each of the following situations:

- Combined CBQ and channel-state dependent packet scheduling (CSDPS) are employed
- Only CSDPS is employed
- Only CBQ is employed
- Only a plain round robin scheduler

We investigate along two dimensions - the effect of changing the traffic types and the effect of changing the loss characteristics of the links. In the first case we used a specific lossy channel and each of the following traffic sources:

- only CBR traffic
- only Poisson traffic
- only Bursty traffic
- a mix of the above traffic types

We present here only the graphs obtained for the first and the last case, and only comment on the remaining cases.

A lossy channel was chosen to magnify the performance improvement for the cases which use channel state dependent scheduling. For the second set of simulations, we fixed the traffic (a mix of the three traffic types, presented above in the three graph) and study the effect of changing channel characteristics.

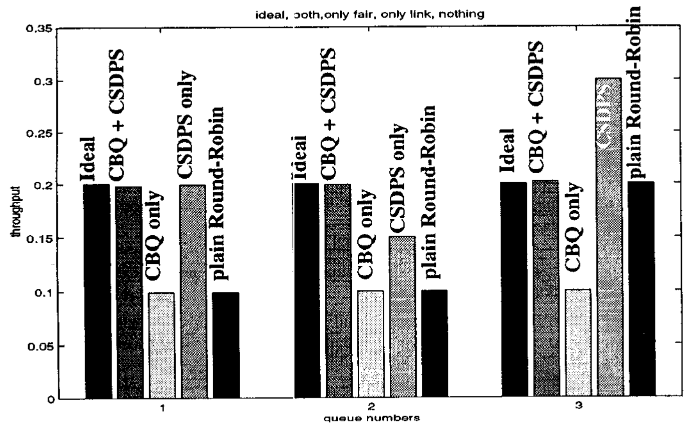

Figure 10: Throughput when all three channels have an offset of one third of their good period duration between their entering the different states, and application three is misbehaving

bandwidth	100 Kbps
slot	10 μs
total simulation time	10 seconds
w_q (average weight of queue length)	0.1
threshold (for backlog)	2 packets
RTS-CTS delay	20 slots

Table 1: Parameters used for simulations

Queue #	0	1	2	3	4	5	6	7
Interarrival	1000	4000	836	1200	800	2000	2000	1600
Packet size	60	80	100	300	80	300	200	400

Table 2: CBR and Poisson traffic parameters

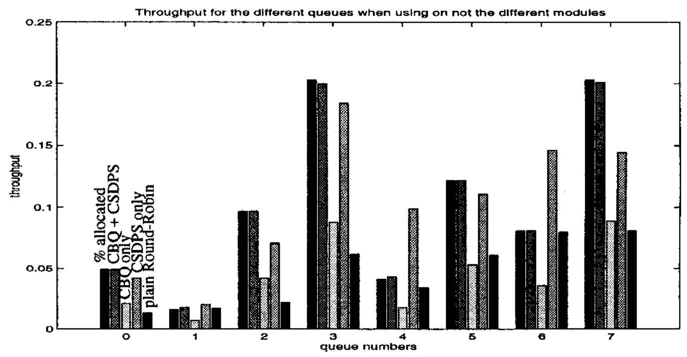

Figure 11: Throughput for mixed traffic and relatively good links

Applications 4 and 6 are made to generate traffic at a rate higher than their share (by making their packet sizes larger) to study the fairness properties of the algorithms.We use a round-robin scheduler that takes the CBQ state and channel-state information into account for the scheduling. Each link is assumed to behave like a two-state Markov chain with a bit error probability of 0 in the good state and 1 in the bad state. The RTS-CTS overhead is 20 bytes per handshake.

The simulations were done using the Maisie discrete event simulator [Short95]. The data rate of the wireless channel was 800 Kbps, and a simulation time slot was chosen to be 10 μs corresponding to the transmission time of one byte.

Table 1 lists the various values we use for our simulations. For the first set of results, with the two state Markov modules used to simulate the links, the probability of transition from the good to bad state in each slot was set to $0.5x10^{-5}$ and the transition probability from bad to good state to 10^{-4}. These numbers corresponds to an average fade length of about 10,000 slots (10 ms), with average fades separated by 30,000 slots (30 ms). This is a high loss link.

Figure 11 shows the throughput when all sources generate CBR traffic with packet sizes and rates as indicated in Figure 2. In the graph the first bar corresponds to the allocated percentage of the achieved bandwidth (bytes/slot), the second to the CBQ+CSDPS scheme, the third to the CBQ scheme (no CSDPS), the fourth to the CSDPS scheme (no CBQ), and the fifth to neither CBQ nor CSDPS (scheduler alone).

The following observations can be made from Figure 11:

- Throughputs are very low if the link-scheduler is not used. This is expected since the link has a high error rate and this causes long head-of-line blocking delays.
- CBQ reduces the overall throughput a bit, but enforces fairness; for example, we observe that queue 6, which is transmitting at above it's allocation, is restricted to stay within its allocation by the use of CBQ since the other queues are unsatisfied. On the

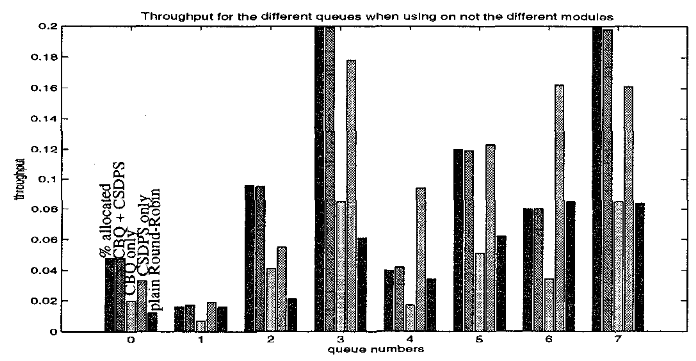

Figure 12: Throughput for mixed traffic and relatively bad links

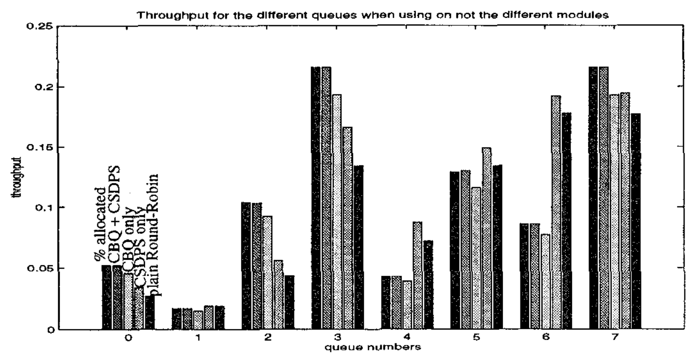

Figure 13: Throughput for mixed traffic and relatively good links

other hand, when CBQ is not used, we see that 6 is getting an unfair portion of the bandwidth thus reducing the throughput of the other queues.

- Adding the CSDPS drastically affects the throughput.
- CSDPS improves delays drastically (the delay graphs are not presented because no special effort was made for delay improvement, it just occurred as a side-effect). CBQ reduces delays for non-violating classes (e.g. class 6) while increasing it for the violating ones - exactly as desired.

For Poisson traffic sources we used the same packet sizes and interarrival delays as for the CBR case. The general shape of these graphs is quite similar to the CBR case, though the average delay values are slightly smaller. The overall throughput is about the same.

For bursty traffic the delays were larger and throughputs smaller, since CBQ restricted a source during bursts of large length.

Figure 12 presents the throughput for a mix of traffic sources, with sources 0, 3 and 5 being CBR, 1, 4 and 6 Poisson and 2 and 7 are bursty (with parameters for each source as in their respective tables given above). This mix of traffic source types represents a more real-world scenario. The throughput performance is observed to be slightly better than the other cases since it is less deterministic and this heterogeneity allows better link sharing. From the above observations we conclude that our modified CBQ behaves more or less fairly towards all types of traffic sources.

Figure 13 shows the effect of improving the link error characteristics. The probability of transition from the good to bad state in each slot is 10^{-4} and the transition probability from bad to good state 10^{-3}. This corresponds to an average fade length of about 1000 slots (1 ms), with fades separated by 10,000 slots (10 ms).

As expected, since the links are really good, the introduction of the channel state dependent packet scheduler does not significantly improve the performance. On the other hand, CBQ does indeed ensure fairness and restricts the misbehaving sources. The overall throughput of the link does approach its optimal value of close to 85% (i.e. no errors - only overhead incurred are due to the RTS-CTS exchange). Also, the effect of CBQ on delay performance of is more pronounced - it drastically increases for the violating classes and decreases appreciably for the non-violating ones.

8.0 Conclusions

Maintaining a high channel utilization (throughput) and ensuring distribution of bandwidth to different connections according to their allocations (fairness) are major requirements in transporting real-time multimedia traffic over wireless links. The challenge is made hard not only by the shared nature of the wireless links but also by the possibility of the links from a transmitter to spatially distributed receivers being in different states. We have described how combining Class Based Queuing and Channel State Dependent Packet Scheduling can enable controlled wireless link sharing. However, we also saw that the basic CBQ and CSDSP needed to be modified to work well with wireless links instead of a naive combination of the two. The twin objectives of throughput and fairness can conflict: throughput might have to be sacrificed for fairness or vice-versa. The time frame of interest over which to provide fairness needs to be selected carefully. Finally, the impact on buffer allocation at the transmitter needs to be studied since a side effect of our approach is that the burstiness of one connection may be modulated by fading and interference affecting another connection.

9.0 References

[Agrawal96] P. Agrawal, E. Hyden, P. Krzyzanowski, P. Mishra, M. B. Srivastava, and J. A. Trotter, "SWAN: a mobile multimedia wireless network," IEEE Personal Communications, vol.3, no.2, pp. 18-33, April 1996.

[Bhagwat96] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. Tripathi, "Enhancing throughput over wireless LANs using Channel State Dependent Packet Scheduling," Proceedings of IEEE Infocom 1996, vol. 3, pp. 1133-1140, March 1996.

[Chen94] K-C Chen, "Medium Access Control of Wireless LANs for Mobile Computing," IEEE Network Magazine, vol. 8, no. 5, pp. 50-63, September/October 1994.

[Floyd95] S. Floyd, and V. Jacobson, "Link-sharing and resource management models for packet networks," IEEE/ACM Trans. on Networking, vol.3, no.4, pp. 365-86, August 1995.

[Karn90] Phil Karn, "MACA - a new channel access method for packet radio," in ARRL/CRRL Amateur Radio 9th Computer Networking Conference, April 1990.

[Karol95] M. J. Karol, Zhao Liu, and K. Y. Eng, "Distributed-queueing request update multiple access (DQRUMA) for wireless packet (ATM) networks," Proc. of the 1995 IEEE International Conf. on Communications (ICC'95), vol. 2, pp. 1224-1231, June 1995.

[Short95] J. Short, R. Bagrodia, and L. Kleinrock, "Mobile wireless network system simulation," Wireless Networks, vol.1, no. 4, pp. 451-467, Baltzer, 1995.

[Sivalingam97] K. M. Sivalingam, M. B. Srivastava, P. Agrawal, "Low Power Link and Access Protocols for Wireless Multimedia Networks," In the Proceedings of IEEE Vehicular Technology Conference, Phoenix, AZ, May 4-7, 1997.

[Swarts94] F. Swarts, and H Ferreira, "Markov characterization of digital fading mobile VHF channels," IEEE Trans. on Vehicular Technology, vol. 43, no. 4, pp. 977-985, Nov. 1994.

[Wang95] H. Wang, and N. Moayeri, "Finite State Markov Channel - a useful model for radio communications," IEEE Trans. on Vehicular Technology, vol. 44, no. 1, pp. 163-171, Feb. 1995.