Fair Scheduling in Broadcast Environments

Nitin H. Vaidya **Paramvir Bahl**

August 1999*

Technical Report
MSR-TR-99-61

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

* Revised December 1999

Abstract

This report considers the problems of scheduling transmissions in broadcast environments, including, wireless environments. Issues that affect the design of fair scheduling algorithms, and several alternative approaches to implementing fair scheduling in single-hop and multi-hop environments are identified.

1 Fair Queuing

In recent years, much research has been performed on the subject of packet fair queuing (PFQ) [2], [3], [9], [12], [22], [23], [26]. Consider the system shown in Figure 1 where a node maintains several *queues* (*flows*[1]), which store packets to be transmitted on an output link. A fair queuing algorithm is used to determine which queue to serve next, so as to satisfy a certain fairness criterion[2].

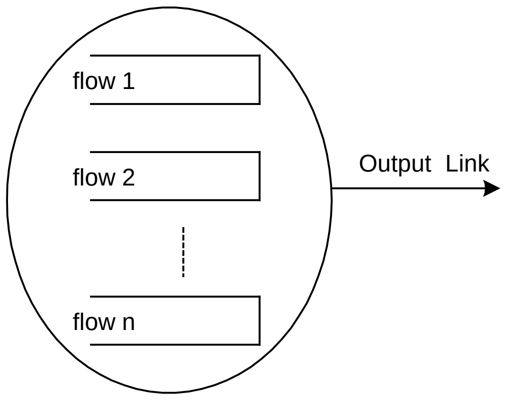


Figure 1: A node with several flows (queues) and a shared link

The fair queuing algorithms in literature attempt to approximate the generalized processor sharing (GPS) discipline [22], [23]. A GPS server serving *n* flows is characterized by *n* positive *weights*; weight w_i being associated with flow i $(i = 1, \ldots, n)$. The GPS server is work-conserving[3]. Let $W_i(t_i, t_2)$ be the amount of traffic served from flow i in the interval $[t_1, t_2]$. Then, for a GPS server [22], if flow i is backlogged[4] throughout $[t_1, t_2]$, the following condition holds:

$$\frac{W_i(t_1,t_2)}{W_j(t_1,t_2)} \geq \frac{w_i}{w_j} \qquad j = 1, \ldots, n \tag{1}$$

Two possibilities exist for assigning weight to each flow:

- A fixed weight is assigned to each flow, potentially based on its priority or bandwidth requirements.

- A weight may be calculated dynamically, potentially using a different weight for each packet in a flow. Such an approach may be useful when bandwidth requirement of a flow is hard to characterize due to significant dynamic variations. In such a case, for instance, the weight of a flow may be chosen to be proportional to its "current" bandwidth

[1] We will refer to each of these queues as a *flows.*
[2] Packets in a given queue are serviced in a first-in-first-out order
[3] A work-conserving server does not idle if there are packets pending to be served.
[4] A queue (or flow) is said to be backlogged if it is not empty.

requirement. With time-varying weights, the condition in Equation 1 would be evaluated only over intervals $[t_1, t_2]$ over which both w_i and w_j are both constant.

Unless stated otherwise, we will assume that the weight of the flow does not vary with time.

The GPS server can interleave packets from different flows with an arbitrarily fine granularity. As shown in Figure 2, a GPS server can, in theory, be implemented using a round-robin scheduler that serves the backlogged session such that the amount of traffic served from a backlogged sessions i in one round is proportional to w_i (where the proportionality constant can be made as close to 0 as necessary). This round-robin algorithm may be described as follows - during each *round* of the algorithm [13], the following steps are performed. The round number r increments by 1 at the end of each round.

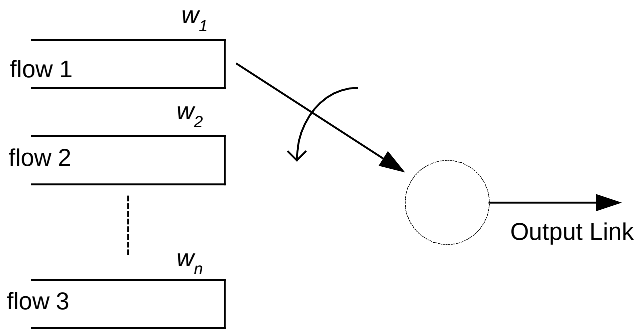


Figure 2: Round-robin server

Algorithm GPS-1

```
For  i = 1 to n
   if  flow i is backlogged   then {
         schedule flow i for transmission for duration $w_i\delta$ where $\delta$ is a constant
   } else
         do not schedule flow i for transmission in this round
```

The above algorithm becomes identical to a GPS scheduler when $\delta \rightarrow 0$.

We now present an alternative implementation of a GPS scheduler. This alternative implementation will later be extended to design a similar idealized scheduler for a multi-hop wireless environment.

Assume that each weight w_i is an integral multiple of some number α. With each flow, associate a timestamp *(round, work)*. Initially, for flow i, the timestamp is $(0, w_i)$. During each round, the following steps are performed:

Algorithm GPS-2

1. Sort the *n* flows in a non-decreasing order of their associated round number (*r)* with ties being broken arbitrarily (i.e. flows with equal round numbers can appear in any relative

order). Let $f_1, f_2, \ldots, f_n$ denote the sorted order with f_1 denoting the flow with the smallest round number. Let (r_i, ϕ_i) denote the time-stamp for flow f_i.

```
2. if flow $f_1$ is backlogged then
       Schedule flow $f_1$ for transmission for duration $\delta$

3. Update its associated timestamp $(r_1, \phi_1)$ as follows
       a. $\phi_1 = \phi_1 - \alpha$
       b. if $(\phi_1 == 0)$ {
              $r_1 = r_1 + 1$;        // increment round number
              reset $\phi_1$ to $f_i$'s original weight;
          }

4. Update the sorted list of flows to account for the new time-stamp of flow $f_1$.

5. Go to step 2
```

The GPS-2 algorithm schedules one flow for transmission for δ time units at a time. Note that in step 3 of the above algorithm, if flow *i* is not scheduled for transmission, then it does not consume any time on the output link. When $\delta/\alpha \to 0$, the above algorithm becomes equivalent to GPS. The round number *r* in the above algorithm is equivalent to *virtual time*, maintained in many fair scheduling algorithm (1 unit of virtual time is required to transmit $w_i(\delta/\alpha)$ units of flow i for some constant α)[5].

Observe that algorithm GPS-1 transmits a backlogged flow *i* for duration proportional to w_i during each turn of the flow. On the other hand, during each turn of a flow, algorithm GPS-2 transmits flow *i* for a duration δ of time, requiring w_i/α turns to transmit a backlogged flow *i* for a duration proportional to w_i. However, both algorithms ensure that a flow *i* does not proceed with round $r+1$ of its transmission while some other backlogged flow has not completed round *r*.

This idealized version of GPS cannot be implemented in practice, since packets must be transmitted as a whole. This observation has motivated many researchers to develop PFQ algorithms, including the popular *Weighted Fair Queuing* (WFQ) [6], also known as *Packet Generalized Processor Sharing* (PGPS) discipline [22].

The work on fair queuing has implicitly assumed that the state of all flows (or queues) is visible to the server. The implicit assumption arises from the fact that the proposed algorithms are designed primarily for the case when a router needs to decide which packet from the local queues to transmit on its output link. This common assumption needs to be modified when considering a broadcast medium.

[5] It might be easier to understand the above algorithm under the assumption that all w_i's are integers and that $\alpha = 1$. The reason for introducing α is to allow arbitrary weights, while approximating GPS with arbitrary accuracy

2 Broadcast Environments

The environment considered in this paper consists of multiple nodes sharing a broadcast medium. The objective is to provide fair distribution of bandwidth amongst these nodes by using an appropriate scheduling (or medium access control) algorithm. Therefore, hereafter, we will refer to the problem under discussion as *fair scheduling*, instead of fair queuing. We consider three broadcast environments with an increasing degree of difficulty in achieving fairness: (1) Wired local area network, (2) Wireless local area network, and (3) Multi-hop wireless network

In the rest of this section, we discuss new issues raised by these broadcast environments.

2.1 Wired Local Area Network

We assume that in a wired local area network (LAN), all nodes can communicate with all other nodes on the LAN, and that transmission errors are negligible. Consider the system illustrated in Figure 3(a). For simplicity, assume that the packets queued at node *i* belong to a single queue (or flow), and that the flow at node *i* is assigned the weight w_i [6].

Observe that a wired LAN can carry a packet by any one flow at any given instant of time. Thus, the job of a fair scheduler (or, a fair medium access control protocol) is to schedule packet transmissions so as to fairly allocate bandwidth to the flows. The LAN shown in Figure 3(a) may be represented as shown in Figure 3(b) which is analogous to Figure 1, where each "flow" in Figure 3(b) represents a node on the LAN, the output link represents the shared broadcast channel, and the node in Figure 3(b) represents the fair scheduler. Thus, it is apparent that the definition of fairness for sharing of a point-to-point link may be applied to transmission scheduling on wired LANs as well[7].

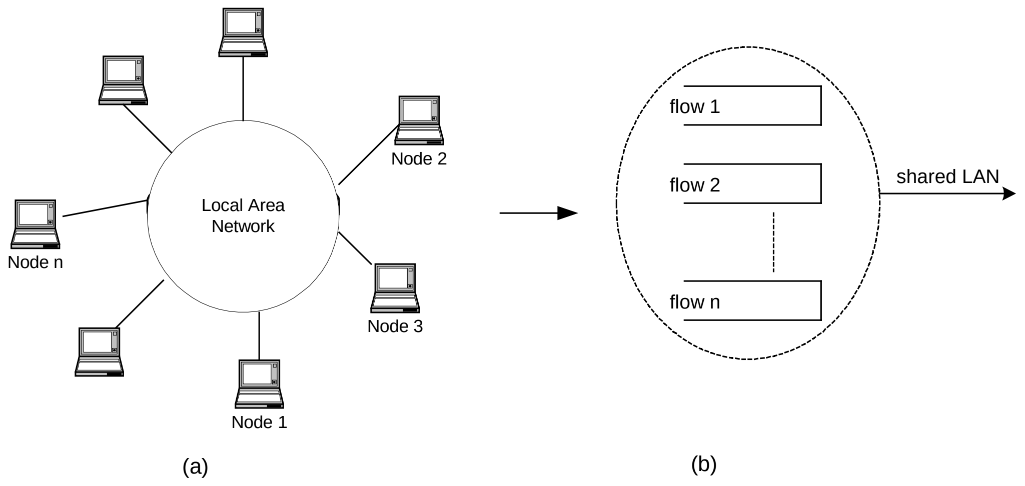


Figure 3: (a) a local area network. (b) A representation of a LAN

[6] In general, multiple queues may be maintained at each node.

[7] When multiple flows are maintained at each node, these flows can all be considered to be at the same ''level", or alternatively, considered a second level in a hierarchy (similar to hierarchical fair queueing [3] where each node represents the first level of the hierarchy).

There are some differences between fair scheduling on a point-to-point link as in Figure 1 and that on a LAN as in Figure 3:

- In case of Figure 1, an algorithm executing at the node decides which packet to transmit next. This algorithm does not need to use the output link for the purpose of decision-making; only the local computing resource (CPU) is used. On the other hand, the scheduling decision on a LAN is a function of the actions taken by all nodes on that LAN (i.e., as governed by the MAC protocol). As such, the process of deciding which packet (or which node) gets to transmit may itself use the broadcast channel.

- As noted earlier, in Figure 1, the fair scheduling server has complete information about the contents of the backlogged queues. However, in a LAN, ordinarily, each node only knows the contents of its local queues. Information about the contents of the different queues can be disseminated between different nodes, however, this consumes the very resource that is to be shared fairly. Thus, in a broadcast environment, a trade-off may exist between *fairness* of a scheduler and the *overhead* (i.e., use of broadcast channel to achieve fairness) [4].

2.2 Wireless Local Area Network

We assume that in a wireless LAN, all nodes can communicate with each other (wireless environments where this is not true will be considered in the context of wireless multi-hop networks).

In addition to the issues raised in the context of wired LAN, a wireless LAN gives rise to two additional problems:

- Transmission errors: Control and data packets may be corrupted and dropped due to wireless transmission errors.

- Wireless transceivers are typically half-duplex, and unable to detect collisions. Therefore, algorithms designed for wired LAN cannot be used, if collision detection is required.

Several algorithms have been proposed that attempt to achieve some notion of fairness in wireless environments despite transmission errors [4], [17], [18], [19], [20], [21], [24]. The schemes proposed consider an infrastructure based network topology where wireless hosts communicate with a base station (access point / control point), but not necessarily with each other. The base station coordinates scheduling on both the uplink and the downlink channels. These schemes may be applied in an ad-hoc LAN environment by selecting a host to act as a coordinator (acting as a base station).

Two issues have been addressed in designing these schemes:

- How to define fairness in presence of transmission errors and design of algorithms to achieve such a fairness.

- How to provide the base station the information regarding queues at the wireless hosts. As noted in the previous section, this information is not readily available, and must be communicated to the base station.

If the information on the backlogged queues at wireless hosts is made available to the base station, the problem of fair scheduling in a infra-structure based wireless network becomes similar to that of scheduling on a wired link and existing centralized algorithms for scheduling on wired link can be adapted as illustrated by the past work [4], [17], [18], [19], [20], [21], [24]. To our knowledge, past work on fair scheduling in wireless networks considers such centralized schemes only. Although reference [4] alludes to a trade-off between centralized versus distributed scheduling, it does not discuss the issue further.

2.3 Multi-Hop Wireless Networks

We use the term *multi-hop wireless network* to refer to any environment where every node is not able to communicate with every other node. For brevity, we also use the term *multi-hop network* to refer to a *multi-hop wireless network*. This section discusses the design issues related to developing a fair scheduling algorithm in multi-hop networks. With the exception of [33], we are unaware of other work in this area. Reference [33] considers a related problem of assigning time slots to links in a multihop network such that the number of time slots assigned to a link in a cyclic schedule is proportional to its weight. However, [33] does not consider the possibility that some links (or, more generally, flows) may not be backlogged at a give time. The fair scheduling problem considered here requires that, at any time, bandwidth be allocated in proportion of weights among only the backlogged flows.

2.3.1 How to specify a flow?

In the discussion above, we assumed that the packets from each flow are serviced in the first-in-first-out order. Also, the decision-making mechanism does not need to know the next immediate destination of a transmitted packet. In case of a wired link, all packets have the same immediate destination, therefore, there is no need to take the destination into account. In case of a LAN, transmission of a packet *P* prevents transmission of any other packet, independent of which node (or nodes, in case of a multicast) packet *P* is being transmitted to. Therefore, in the absence of transmission errors, the decision-making mechanism does not need to know the immediate destination of packet *P* on the LAN. Note that in wireless environments, location-dependent errors can occur, making it useful to take the destination of a packet into account [4], [21]. In our current discussion, we do not consider the issue of transmission errors. Even in the absence of transmission errors, in case of multi-hop wireless networks, it is useful to consider the destination of a packet. For instance, Figure 4 shows a network of 5 nodes - a link between a pair of nodes indicates that these nodes can communicate with each other. In Figure 4, whether simultaneous transmissions from nodes B and C will interfere at the receiving nodes or not, depends on who the intended receivers are. If transmissions from nodes B and C are intended for nodes A and E, respectively, then the transmissions will interfere at receiver E. If, on the other hand, transmissions from nodes B and C are intended for nodes A and D, then these simultaneous transmissions may not interfere at the receiving nodes.

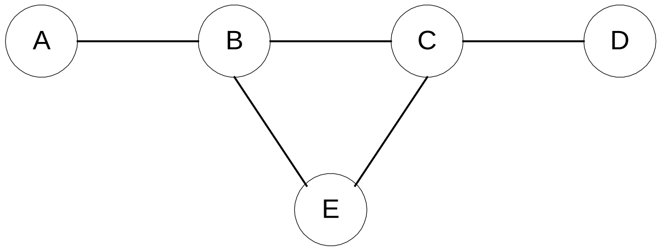


Figure 4: Example: How to define a flow?

The above discussion suggests two approaches to organizing a ”flow” or a ”queue” at a node in a multi-hop network.

1 *Approach 1:* The immediate destination of each packet in a single flow must be identical. Multiple flows may exist at each node.

2 *Approach 2:* Packets within a single flow may be destined to different neighbors. This leads to undesirable situations, as illustrated next. Consider Figure 4. Assume that in a given flow (queue) at node C, the packet at the front of the queue is destined to node E, and the second packet is destined to node D. Now, assume that node B is transmitting a packet to node A. In this case, although node C is unable to transmit the packet at the front of the queue (packet destined to node E), it can potentially transmit the second packet (destined to node D). Although transmitting the second packet first would result in a higher "utilization" of the wireless medium[8], it results in a non-FIFO service order.

In the rest of our discussion on multi-hop wireless networks, we assume that multiple flows may be maintained at each node, however, immediate destination of all packets within a given flow must be identical.

2.3.2 Conflicting Flows and Conflict Graphs

Definition 1: *Flows f_1 and f_2 are said to conflict with each other if packets from these two flows cannot be scheduled for transmission simultaneously. Two flows are said to be conflict-free if they do not conflict with each other.*

Whether transmissions from two flows will conflict or not depends on the physical position of the sender and destination nodes, the transmission range, the MAC protocol they are using (e.g., does the MAC protocol require the destination to send an acknowledgement or not), whether directional antennas are used or omni-directional antennas are used [15], etc.

Definition 2: *A conflict graph $G = (V,E)$ is defined such that V is the set of all flows, and an edge (f_i,f_j) belongs to E if and only if flows f_i and f_j conflict with each other.*

2.3.3 Probabilistic Protocols Cannot Achieve ’’Fairness’’

In this section, we argue that ‘‘probabilistic” algorithms cannot achieve fairness in multi-hop networks. As yet, we have not defined what we mean by *fairness* in the context of multi-hop networks. However, to make the argument, in this section, we will consider a simple scenario in which it is possible to intuitively characterize what fairness should mean. In subsequent sections, we discuss how fairness in multi-hop networks may be defined in general. To explain what we mean by a *probabilistic* protocol, consider the scenario depicted in Figure 5. Four single-hop connections are established in this example. Presence (absence) of a link between two nodes indicates that the two nodes can (cannot) hear each other’s transmission. Arrows in the figure show the direction of transfer for each connection, and its endpoints.

[8] Later in the paper, we extend the notion of a *work-conserving schedule* to formalize what we mean by maximizing utilization in a multi-hop network.

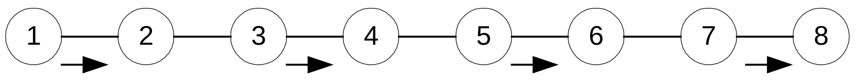


Figure 5: An example scenario

Packets from each connection form a single flow at the originating node for that connection. In this scenario, MAC protocols such as IEEE 802.11 [10] and HomeRF SWAP-CA [30] would allow simultaneous transfers on the following combinations of connections: (a) 1→ 2 and 5→ 6, (b) 3→ 4 and 7→ 8, and (c) 1→ 2 and 7→ 8. Other combinations are disallowed since they may result in interference between data and/or ACK packets. In our discussion in this section, we assume that only the above combinations of transfers are permitted to occur simultaneously by the protocols under consideration (i.e., only these three pairs of flows are conflict-free). Thus, the conflict graph for the scenario under consideration is as shown in Figure 6.

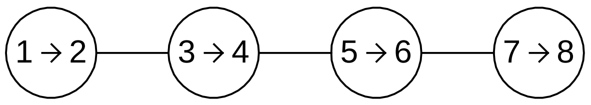


Figure 6: Conflict graph for the example scenario in Figure 5

Deterministic protocols: A deterministic protocol consists of a set of deterministic rules - an example of such a rule is as follows: if flows 1→ 2 and 3→ 4 are both backlogged, then the two flows transmit packets alternatingly. Thus, if flow 1→ 2 just finished transmitting a packet, then flow 3→ 4 is guaranteed to be able to send a packet before flow 1→ 2 will transmit another packet.

Probabilistic protocols: The IEEE 802.11 and HomeRF SWAP-CA medium access control protocols are examples of a probabilistic protocol. A probabilistic protocol cannot guarantee a certain deterministic ordering on the transfers performed by different nodes. When using a probabilistic protocol, *if two nodes are contending to perform a packet transmission, either one of them may ''win'' with a non-zero probability*. A deterministic protocol can make this probability zero for some nodes if desired. For instance, in the example of the deterministic rule discussed above, the probability of node 1 transmitting a packet to node 2, immediately after performing such a transfer is 0, if node 3 has a packet to be sent to node 4.

Clearly, implementation of a deterministic protocol would require *a priori* coordination (to agree on the deterministic ordering) between nodes that may or may not be within each other's transmission range. Many probabilistic protocols, such as 802.11, are fully distributed, and require no *a priori* coordination between nodes contending for data transfer. The lack of the need for *a priori* coordination makes distributed probabilistic protocols attractive. However, we now show that such protocols cannot achieve ''fairness''.

Again, consider the scenario in Figure 5. Assume that the four flows have identical weights. Also assume that the four connections are always backlogged, and that data packets are of a constant size. Under these circumstances, it is possible to schedule packet transmissions such that all flows receive equal bandwidth - such an allocation may be considered ''fair'' since the weights of the flows are equal. Such a fair allocation is achieved by allowing the following two sets of flows to transmit packets with equal frequency (flows in each set transmit packets at the same time, since they do not conflict): (a) 1→ 2 and 5→ 6 and (b) 3→ 4 and 7→ 8. Importantly, the combination of flows 1→ 2 and 7 → 8 should never be scheduled simultaneously.

Although the above schedule may be considered fair, we now argue that such fairness is difficult (if not impossible) to achieve using probabilistic protocols. To simplify, let us assume that the MAC protocol proceeds in ‘‘rounds” - in each round, all nodes wishing to transmit a packet contend for access; a subset of these nodes wins, and transmits packets. As noted before, only three combinations of transfers may occur simultaneously: (a) 1→ 2 and 5→ 6 and (b) 3→ 4 and 7→ 8. (c) 1→ 2 and 7→ 8. Since all flows are backlogged, it follows that the probabilistic protocol will allow one of these three sets of transfers to occur in a given round. Since the algorithm is probabilistic, it will allow, with a non-zero probability, the connections in set (c) to transmit simultaneously. It is easy to see that, regardless of how often sets (a) and (b) transmit, if set (c) can transmit with a non-zero (or non-negligible) probability, the bandwidth allocated to the four connections cannot possibly be equal.

The above argument implies that a probabilistic algorithm cannot achieve fair allocation of bandwidth. This result was shown under two assumptions: (i) we implicitly assume that if a backlogged node can transmit a packet without conflicting with other transmissions, then it will transmit a packet, and (ii) the protocol proceeds in rounds. The first assumption is reasonable since violating that assumption would imply that bandwidth is unnecessarily wasted. The second assumption is often not true, however, we believe that our conclusion is correct even when the protocol does not proceed in rounds. For instance, we simulated the environment in Figure 5 using the IEEE 802.11 MAC protocol assuming a 2 Mbps wireless channel (actual achievable throughput is about 1.3 Mbps, due to header and other protocol overhead). The average throughputs obtained for the four flows were as follows:[9]

Flow	1 → 2	3 → 4	5 → 6	7 → 8
Throughput (Kbps)	85.3	43.4	42.4	86.4

Short-term fairness versus long-term fairness: Consider a scheduling algorithm that allows flows 1→ 2 and 7→ 8 to transmit simultaneously for a small period of time at the start of an interval when both flows are backlogged, however, does not schedule them simultaneously after some time into such an interval. Over a long backlogged interval, such a scheduler could result in a fair bandwidth allocation. However, if duration of a backlogged interval is short, such a scheduler could result in an unfair allocation of bandwidth among the backlogged flows. To put it differently, even if an algorithm can ‘‘learn” to schedule fairly after some duration of unfairness, if the backlogged intervals are of a relatively short duration, the unfairness caused during the learning period would result in significant deviation from a fair allocation of bandwidth. (With reference to the above example, an algorithm that may initially allow combination (c) to transmit simultaneously, but disallows it later on could be fair over a long interval. However, if the backlogged intervals are short in duration, this initial unfairness may result in disproportionate bandwidth allocation to flows 1→ 2 and 7 → 8.)

[9] Note that the actual throughput values may come out different if simulation parameters such as carrier sense power threshold are varied. In any event, the throughputs for the four flows are not obtained to be equal to each other. Incidentally, observe that the two connections in the middle have half as much throughput as the two edge connections. The reason being that, with our model parameters, the probability of each of the three combinations of flows ‘‘winning” is identical. This may change when parameters are changed. The results reported in this table are obtained using parameters used in the IEEE 802.11 model distributed with the *ns* simulator.

2.3.4 Work-Conserving Schedules for Multi-Hop Wireless Networks

In the discussion here, let us consider an ‘‘omniscient” scheduler (server) for a multi-hop network that somehow has information regarding the flows at all the different nodes in the network. This scheduler can determine which packets should be transmitted, and communicate this information to the different nodes, without using any of the wireless resource to be shared fairly. The motivation behind this is to determine how fair scheduling may be performed in an ideal setting, so that practical algorithms may be designed to approximate the ideal.

The notion of a work-conserving server has been defined for the case of a server that provides service on a single link [13]. Such a work-conserving server never remains idle if any of the flows sharing the link is backlogged. In other words, a work-conserving server maximizes the service provided, and, for a given set of flows, no other server may provide more service than a work-conserving server. It follows that, although multiple work-conserving scheduling algorithms can be designed, the busy periods of all the work-conserving algorithms, during which at least one flow is backlogged, are identical [2].

We now extend the notion of work-conserving servers to scheduling in multi-hop wireless networks.

Definition 3: *A scheduling discipline is said to be work-conserving if it schedules non-conflicting flows for simultaneous transmission such that additional simultaneous transmission from any other backlogged flow would result in a conflict.*

As noted above, in the case of multiple flows sharing a single link, all work-conserving schedules would result in the same busy times, i.e., the intervals of time when all flows are empty would be identical for all work-conserving schedules. However, in the case of multi-hop wireless networks, this is not true, as we now illustrate using examples.

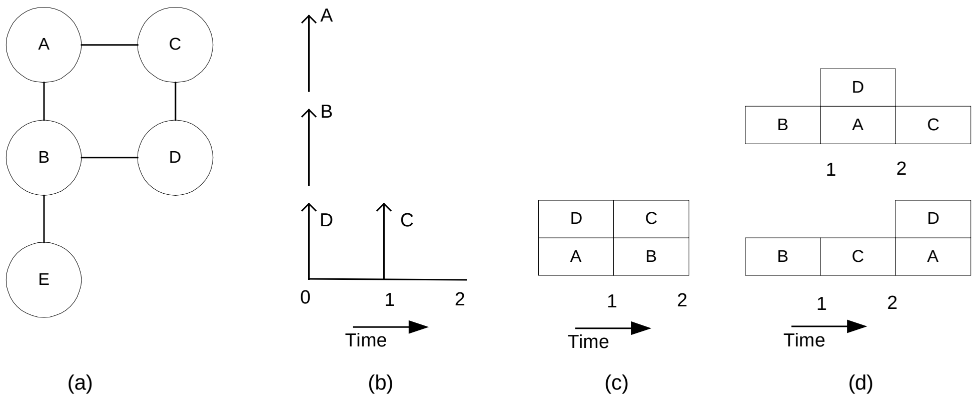


Figure 7: Busy periods of different work-conserving schedules

Figure 7(a) shows conflict graph for a system that contains 5 flows - recall that a link between a pair of flows in a conflict graph indicates that they conflict with each other. Figure 7(b) shows arrival times of packets on flows A, B, C and D - assume that the time required to transmit each packet is 1 time unit. Figures 7(c) and (d) show outcome of three work-conserving schedules. The busy periods for the schedules in figures (c) and (d) are different. From a practical standpoint, a work-conserving schedule with a shorter busy period is better, since it makes more efficient use of the bandwidth - for instance, the schedule in Figure 7(c) may be preferable over those in Figure 7(d). Unfortunately, at a given time, it is not always possible to determine which schedule would lead to the shortest busy period, without having knowledge about future packet arrivals. This is demonstrated in Figure 8.

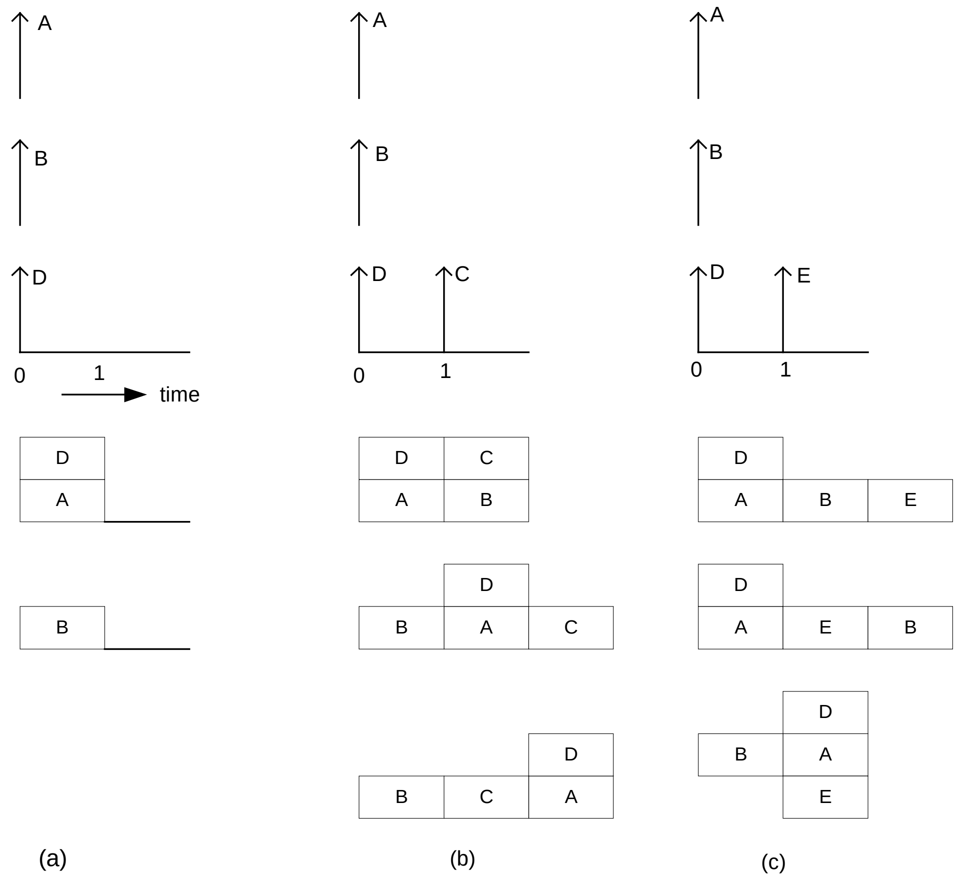


Figure 8: Implementing work-conserving schedules with shortest busy periods requires knowledge of the future

For the example illustrated in Figure 8 assume the same conflict graph as in Figure 7(a). In Figure 8(a), at time 0, one packet each arrives on flows A, B and D, and no other flows are backlogged (each packet is again assumed to require 1 time unit transmission time). A work-conserving server has two alternatives: schedule the packet in flow (or queue) B for transmission at time 0, or alternatively schedule packets in flows A and D at time 0, both of which are shown in Figure 8(a). Now in Figure 8(b), a packet arrives in flow C at time 1. Continuing with the two choices in Figure 8(a), we can now obtain three possible schedules as shown in Figure 8(b) - note that these three schedules are identical to those in Figure 7. Now, if instead of a packet arriving on flow C at time 1, a packet arrives on flow E at time 1, we get the three work-conserving schedules shown in Figure 8(c). Now observe that the packets scheduled for transmission at time 0 in the shortest schedules in Figure 8(c) and (d) are different. However, knowing which of the two possibilities to choose depends on which packet arrives at time 1; if this information is not available at time 0, the work-conserving scheduler cannot guarantee a minimal length schedule.

2.3.5 Is Existing Definition of Fairness Useful?

In the context of multi-hop wireless networks, the definition of fairness, as presented in Section 1 is not useful. As per this definition, ideally, the goal of a fair scheduler should be to schedule transmissions such that, if flows *i* and *j* with weights w_i and w_j, respectively, are *both*, backlogged over interval $[t_1, t_2]$, then

$$\frac{W_i(t_1,t_2)}{W_j(t_1,t_2)} = \frac{w_i}{w_j} \quad j = 1,\ldots,n \tag{2}$$

where, as defined earlier, $W_i(t_1,t_2)$ is the amount of flow *i* traffic served in the interval $[t_1, t_2]$. This definition of fairness is usually not enforceable using a work-conserving scheduler for multi-hop networks, even if an idealized scheduling algorithm is assumed (similar to GPS). As an example, consider the conflict graph for five flows shown in Figure 9. Note that the weight of each flow is $1/3$. Now consider two cases:

Case 1: All five flows are backlogged. In this case, a transmission schedule does exist that can allocate the five nodes equal bandwidth - since the five flows have equal weights, allocating equal bandwidth to them satisfies Equation 5. To allocate the equal bandwidth, a fair scheduler would serve flow A, then flows B and D together, and then flows C and E together, repeatedly, performing equal quanta of work for them during each transmission.

Case 2: Assume that flow B is not backlogged, but the other four flows are backlogged. In this case, Equation 2 would again suggest that equal bandwidth be allocated to the four backlogged flows, since their weights are identical. However, such an allocation cannot possibly result when using a work-conserving server. To see this, observe that flows A, D and E are mutually conflicting, and flows D and E do not conflict with C. Therefore, the sum of bandwidth allocated to flows A, D and E would be equal to the total available bandwidth *r* (assuming a work-conserving server). Since their weights are equal, by Equation 5, each of these three flows would be allocated bandwidth *r/*3. Since flows A and C have equal weights, this implies that flow C is also assigned bandwidth *r/*3. However, since flow A is assigned only *r/*3, and since flow C only conflicts with flow A, it is possible to allocate bandwidth 2*r/*3 to flow C - in fact, allocating less that 2*r/*3 would result in a schedule that is not work-conserving.

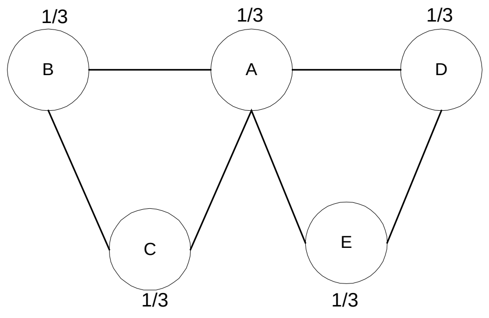


Figure 9: Standard definition of fairness is not useful in multi-hop networks

2.3.6 Definition of Fairness in Multi-Hop Wireless Networks

In the above discussion, we saw that probabilistic protocols may not be able to achieve "fairness" as per an intuitive notion of fairness, and also that the standard definition of fairness may not apply. We did not formally define what fairness means in a multi-hop environment. In this section, we argue the need for a new definition of fairness in multi-hop wireless networks, and then provide one such definition (obtained by extending the GPS definition).

The existing algorithms for fairness on a shared link are designed to approximate the GPS discipline. With these algorithms, at most one flow may be using the shared resource at any given time. In the context of multi-hop wireless networks, the situation gets somewhat more complicated.

In the multi-hop wireless environment, although all hosts may transmit on the same channel, multiple simultaneous transmissions are often possible[10]. Also, the number of simultaneously possible transmissions varies, depending on the relative location of the transmitters. Therefore, it is not always possible to model the multi-hop wireless environment as consisting of a fixed number (one or more) of shared resources. Therefore, a new definition is needed to determine what ‘‘fair" means in the context of multi-hop wireless environments.

In this paper, we take the first step towards defining fairness in multi-hop networks, by defining a *Generalized Resource Sharing* (GRS) algorithm for multi-hop wireless environments. Future work would attempt to mathematically characterize behavior of this (and other similar) algorithm, and identify ways to implement a practical approximation.

Assume that n flows numbered $1, 2, \ldots, n$ are to be scheduled fairly, where w_i is the weight of flow i. We associate a time-stamp (r, ϕ) with each flow, similar to algorithm GPS-2. Initially, the time-stamp for flow i is $(0, w_i)$. Assume that each weight w_i is an integral multiple of some constant α. The server repeatedly performs the following steps:

Algorithm GRS

1. Sort the *n* flows in a non-decreasing order of their associated round number (*r*) with ties being broken arbitrarily (i.e. flows with equal round numbers can appear in any relative order).

   Let $f_1, f_2, \ldots, f_n$ denote the sorted order, f_1 denoting the flow with smallest round number. Let (r_i, φ_i) denote the time-stamp for flow f_i.

2. *Scheduled_Set* = {}

```
for i = 1 to n {
     if  (f_i does not conflict with any flow in Scheduled_Set)
          then {
               if  (flow f_i is backlogged) {
                    schedule flow f_i for transmission for duration $\delta$
                    Scheduled_Set = Scheduled_Set $\cup$ {f_i}
               }
```

[10] When multiple channels are available, the ideas presented in this paper can be readily extended to that case as well.

```
                Update its timestamp $(r_i, \phi_i)$ as follows:
                    a.  $\phi_i = \phi_i - \alpha$
                    b.  $if\ (\phi_i == 0)\ \{$
                           $r_1 = r_1 + 1;$          // increment round number
                           reset $\phi_1$ to $f_i$'s original weight;
                        }

                Update the sorted list of flows to account for the new time-stamp of $f_1$.
            }
    }
```

3. All flows in *Scheduled-Set* transmit simultaneously for duration δ.

4. Go to step 2

The ideal behavior of the algorithm is achieved when δ/α approaches 0. The time-stamp of a flow represents the most recent time when the flow is serviced. Note that the GRS algorithm needs to determine whether a given flow can conflict with the set of flows already scheduled for transmission in a given round. In GPS-2, this step is not needed since any scheduled flow on a wired link conflicts with all other flows.

In fact, the above GRS algorithm becomes equivalent to GPS-2 if we assume that every flow conflicts with every other flow. Our future work will investigate the properties of the above algorithms, and other similar algorithms (for instance, a special case of the above algorithm that sorts flows based on the tuple *(r,ϕ)* or *(r,ϕ/w)* instead of just the round number *r* is of particular interest). We believe that practical algorithms that can approximate the above idealized algorithm can be obtained by using the LAN-based fair scheduling algorithm presented below in a multi-hop environment.

2.3.7 Impact of Asymmetric Links

Two forms of asymmetries can exist:

- Node A can receive packet from node B, but node B cannot receive packet from node A. In presence of such an asymmetry, medium access control (MAC) protocols that use an acknowledgement for reliability (e.g., IEEE 802.11) are not applicable.

- A node using a CSMA protocols needs to be able to sense when another node is transmitting, so that it can defer transmission. A power level threshold, say *Defer-Threshold* is used for this purpose - if receive power level is above the *Defer-Threshold*, the medium is deemed to be busy [11]. Similarly, a node must decide when to attempt to receive a packet, based on signal strength. Again, a power level threshold, say *Receive-Threshold* is used. If the receive power level is above this threshold, only then a node may attempt to decode received transmission. Reference [11] argues that in some environments it makes sense to have a significantly lower *Defer-Threshold*, as compared to the *Receive-Threshold*. With such thresholds, it is possible that two nodes cannot receive each others transmissions (since received power < Receive-Threshold), but would defer to each other (since received power > Defer-Threshold). In a multi-hop environment, however, with such threshold settings, two nodes that must defer to each other cannot communicate with each other. Therefore, any protocol that requires message exchange between nodes that conflict with each other cannot be used. (For instance, the slot reservation protocol in [16] for multi-hop networks cannot be implemented).

3 Fair Scheduling on a Local Area Network

Many approaches for fair scheduling on a LAN are possible. In this section, we summarize several approaches first, followed by evaluation of a distributed contention-based protocol.

3.1 Centralized Schemes

The centralized schemes attempt to emulate a fair scheduling algorithm for a point-to-point link by choosing one node on the LAN as a *coordinator*. We observe that fair queuing algorithms for the point-to-point link (Figure 1) typically only need to know the size and arrival time of packets in each queue (in addition to the weights of all queues). Thus, in the emulation, each node sends to the coordinator information on packets in its local queue. With this, the coordinator will have all the information it needs to apply traditional fair queuing algorithms. Therefore, the coordinator node can determine (using existing fair queuing algorithms) which packet should be transmitted next, and send a "go" message to the corresponding node. On receiving the *go* message, a node can transmit the packet at the front of its queue.

This leaves open the question of how a node may send the information on the packet at the front of its queue. Several different techniques may be used, resulting in different trade-offs (such schemes have been studied previously in the context of fair scheduling in infra-structure based networks [4], [17], [18], [19], [20]) :

- When an empty queue becomes backlogged, a node may use a ‘‘contention-based” access to send the information to the coordinator node (some interval of time would have to be set aside for the contention mode).
- The coordinator node may periodically poll nodes whose queue are known to have been empty, to see if the queues are now backlogged. In this case, the coordinator may not immediately learn when a previously empty queue becomes backlogged, leading to unfair bandwidth allocation until the coordinator learns the true state of the queue.
- When a node transmits a packet, it can piggyback necessary information about the packet it wants to transmit next.
- The coordinator may periodically poll all nodes to receive updates on their queues’ status.

The different techniques above would result in a trade-off between fairness and overhead due to bandwidth used for communicating packet size information to the coordinator node.

3.2 Round Robin

To implement round robin schemes, the nodes on a LAN must be able to organize themselves in a logical cycle. Two forms of round robin algorithms are possible, as discussed below.

3.2.1 Using Round Robin to Rotate Permission to Transmit

This is the common form of round robin mechanism wherein each node gains the right to access the channel after the previous node in the cycle. This mechanism may be implemented in at least two different ways:

- A node explicitly transmits a "token" packet to the next node, when it wishes to give up its turn to transmit.

- Alternatively, time may be divided into "time slots" (similar to time slots in IEEE 802.11), and a node that gets a turn to transmit, may give up its turn by staying silent for one time slot (as opposed to forwarding an explicit token to the next node in the cycle). The next node, seeing silence for one time slot, presumes that it is now its turn to transmit.

The first alternative may be preferred when the likelihood of errors is high or fading occurs often - in this case, a node may erroneously perceive another node as silent even though it is not. Using tokens (and acknowledgements for the same) would alleviate this problem.

The above two alternatives only consider the issue of how to rotate the permission to transmit. The additional issue that must be resolved is that of determining how a node decides whether it should transmit or not during its turn. Clearly, if a node is not backlogged during its turn, it cannot transmit a packet. However, for fair allocation, it may not be possible for a backlogged node to transmit during its given turn.

To determine if a given node should transmit during a given turn, the above round robin mechanism can use any of the round robin fair scheduling algorithm that have been previously proposed for the case of a wired link, for instance, *Weighted Round Robin* and *Deficit Round Robin* [26].

3.2.2 Using Round Robin to Implement Fair Medium Access Control

In the above approach, a node holding the token had permission to access the medium. A fairness algorithm (such as deficit round robin) is used locally at each node to determine whether to transmit during a given turn or not. Due to the distributed nature of decision-making in the above approach, only algorithms that use local state at a node can be applied.

An alternative to the above round robin mechanism is to use a round robin technique to determine who should transmit next, so as to allocate the bandwidth fairly.

The proposed protocol is divided into two phases: a decision phase and a transmission phase. Start of a new decision phase (cycle) is indicated by the end of a packet transmission. The decision phase is implemented using a round robin mechanism. During its turn in the round robin, each node transmits some information, which may be used in determining who gets to transmit in the transmission phase (a node not wishing to transmit such information may either pass a token to the next node, or remain silent, as discussed above). At the end of the decision phase, all nodes can use the available information to decide who should transmit next. In absence of errors, all nodes will have identical information, and reach the same decision.

For instance, the *Start-Time Fair Queuing* [9] algorithm proposed for wired links may be implemented using this approach. We first describe Start-time-Fair Queuing (SFQ).

Start-Time Fair Queuing [9]

Start-time fair queuing (SFQ) assumes system architecture shown in Figure 1. A virtual clock is maintained, and *v(t)* denotes the *virtual* time at physical time *t*. Let P_i^k denote the *k*-th packet arriving on flow *i*. Let A_i^k denote the real time at which packet P_i^k arrives. Let L_i^k denote the size of packet P_i^k. A start tag S_i^k and a finish tag F_i^k are associated with each packet P_i^k, as described below. Initially, $t=0$. $v(0)=0$ and $F_i^0=0, \forall\ i$.

1. On arrival of packet P_i^k at node *i*, the packet is stamped with start tag S_i^k, calculated as

   $$S_i^k = \max\left\{v(A_i^k), F_i^k\right\}$$

   where F_i^k, the finish tag of P_i^k is calculated as

   $$F_i^k = S_i^k + \frac{L_i^k}{w_i}$$

2. Initially, the virtual clock is set to 0, i.e., $v(0)=0$. The virtual time is updated only when a packet is transmitted on the link. If at time *t*, a packet is in service, then *v(t)* is updated to equal start time of that packet.

3. Packets are transmitted on the link in the increasing order of their start tags. Ties are broken arbitrarily.

Implementing SFQ in a LAN using Round Robin Mechanism

To implement the SFQ algorithm, in our round-robin mechanism, each node wishing to transmit a packet would send the start tag for its packet during its turn in the decision phase. At the end of the decision phase, the node with the smallest start tag would transmit, ties being broken using a deterministic rule.

The above procedure addresses the issue of determining which start tag is the smallest. To calculate start tags, SFQ maintains a virtual clock. To maintain virtual time at each node in a distributed manner, each transmitted packet would need to be tagged by its start tag[11].

The overhead of transmitting start tags may be reduced as follows:

- A node does not transmit its start tag if an earlier node in the decision phase has already advertised a start tag that is smaller than (or equal to) its own start tag.
- The overhead could be further reduced if a node keeps track of start tags advertised by nodes (including itself) that have not had an opportunity to transmit yet. In this case as well, a node would not transmit its start tag if any of these recorded start tags is smaller than or equal to its own start tag.
- Finally, the overhead may be further decreased by skipping turns to nodes that have already advertised their start tags but have not had an opportunity to transmit yet.

Note that the last two optimizations are useful only if a node that has advertised a start tag does not drop the packet before it has a chance to transmit.

[11] Alternatively, the start tag may be deduced using information transmitted during the decision phase

3.3 Contention-Based Distributed Protocol

In the centralized and round robin protocols discussed above, contention between traffic from different nodes is avoided either by using a coordinator node or by a round robin mechanism. On the other hand, in the contention-based protocol proposed here, different nodes contend for access to the channel, and, hopefully, the contention mechanism would choose the winner (who could transmit next) such that the bandwidth is allocated fairly to all the nodes.

We are not aware of any work on fully distributed protocols that provide fair scheduling or quality-of-service, with the notable exception of [28]. However, discussion in [28] is restricted to the case of real-time flows that schedules transmissions with a fixed interval between consecutive transmissions. The fair scheduling algorithm presented here does not make such assumptions.

The contention-based protocol is fully distributed, with each node having knowledge of only its local queue. The advantage of such an approach is that (a) there is no need to elect a coordinator, as in a centralized approach, and (b) there is no requirement that the nodes arrange themselves in a cycle. More importantly, a fully distributed approach would be better suited for an environment where nodes join and leave dynamically, and no single node may have information on the ‘‘weights” of flows at different nodes (note that centralized approaches use such information).

3.3.1 Issues in Design of a Contention-Based Protocol

By the term *contention-based* protocols we refer to protocols that do not perform any a priori coordination for performing medium access. Contention-based protocols have been proposed previously for priority-based scheduling, and also for providing quality-of-service [1], [5], [14], [25], [27], [28]. Several variations for contention-based protocols are plausible:

- In Contention-based protocols, before a node transmits a packet it wins the right to transmit. Two general approaches for these are as follows:

    - The node transmits a signal (called black burst in [27], [28]) of a suitably chosen duration. The node with the longest signal wins the right to transmit.

    - A node waits for a suitably chosen interval of time (called backoff interval in IEEE 802.11[10], and if the medium is still idle at the end of this interval, it wins the right to transmit.

  The second of the two approaches may result in lower energy consumption.

- Collision Resolution: In both of the above approaches, it is possible that different nodes may choose the same interval, and thus multiple winners may exist[12]. When these multiple winners attempt to transmit simultaneously, collision would occur. A collision resolution protocol [8], [29] is needed to recover from a collision. Two variations are possible:

    - Nodes that were the winners in the above step get an opportunity to transmit (using the collision resolution protocol) before any other node will attempt to transmit a packet.

[12] By making assumptions about the traffic and the time between consecutive packets, [27], [28] are able to avoid such collisions.

  - The winners, on detecting a collision, reinitiate the first step by choosing a new interval (for backoff or black burst). The procedure for choosing the new interval after collision of data packets may be different from that for choosing the interval for the first time for a given packet.

We consider a contention-based protocol, which is, derived from the Start-Time Fair Queuing scheme [9] described earlier and the IEEE 802.11 [10] (and HomeRF SWAP-CA [30]) MAC protocol. Our protocol differs from 802.11 and SWAP-CA in the way in which the backoff interval is calculated, and updated (before and after collision of data packets). Further work is required to optimize this protocol. Also, in this paper, we only present one implementation, although as alluded above several other variations are possible.

To facilitate discussion of the proposed scheme, we first describe relevant features of the 802.11 MAC protocol (in particular, we will use the Distributed Coordination Function approach in 802.11).

3.3.2 The Distributed Coordination Function of the IEEE 802.11 MAC

When a node *i* wishes to transmit a packet, it chooses a *backoff interval*, B_i: B_i is an integer, and the actual backoff interval is equal to B_i *slots*. Specifically, B_i is uniformly distributed in the interval [0, cw], where *cw* is the size of a *contention window*. *cw* at node *i* is reset to a value CW_{min} at the beginning of time, and also after each successful transmission of a data packet by node *i*.

Now, if the transmission medium is not idle, node *i* waits until it becomes idle. While the medium is idle, B_i is decremented by 1 after each slot time. Actually, the node waits for an interval known as an inter-frame spacing, before starting to decrement B_i. We will omit such minor details in this discussion. However, our simulation model does implement these details accurately. If the medium becomes busy while B_i is non-zero, then B_i is frozen while the medium is busy. B_i is decremented again when the medium becomes idle. Eventually, when B_i reaches 0, node *i* transmits a Request-to-Send (RTS) packet to the intended destination of the packet. The destination node, on reliably receiving the RTS, responds with a Clear-to-Send (CTS) packet. The node *i*, on receipt of the CTS packet, transmits the data packet.

Now, it is possible that two nodes, say *i* and *j*, may choose their backoff intervals such that they both transmit their RTS packets simultaneously, causing a collision between the RTS packets at destination node *i*. In this case, node *i* will not receive a CTS, therefore, it will not be able to send the data packet.

When a CTS is not received, nodes *i* and *j* both double their contention window size *cw*, pick a new B_i uniformly distributed over *[*0,*cw]*, and repeat the above procedure.

3.3.3 Distributed Fair Scheduling -- Proposed Fair Scheduling Protocol for a LAN

The proposed *Distributed Fair Scheduling* (DFS) protocol is based on the 802.11 (also SWAP-CA) MAC and SFQ:

- The proposed MAC protocol borrows on SFQ's idea of transmitting the packet whose start time is smallest, as well as SFQ's mechanism for updating the virtual time. However, our implementation of the virtual time, and the process of determining the smallest start time is distributed.

- The distributed mechanism for determining the smallest start time is based on the idea of a *backoff interval* utilized in 802.11 MAC.

The above two features together allow us to approximately implement SFQ in a distributed manner. We now describe the proposed approach.

Assume that all packets at a node belong to a single queue (the algorithm can be easily extended when multiple queues are maintained at each node). Each node *i* maintains a virtual clock, $v_i(t)$, where $v_i(0)=0$. Start tags and finish tags are calculated as in the SFQ algorithm. Whereas P_i^k represented the *k*-th packet arriving on flow *i* in the discussion of SFQ, now P_i^k represents the *k*-th packet arriving at the flow at node *i* on the LAN.

1. Each transmitted packet is tagged with its start tag.

2. When at time *t* a node *i* hears a packet with start tag *s*, node *i* updates its virtual clock as follows:
$$v_i(t) = \max(v_i(t), s)$$
The virtual clock is not updated at any other time.

3. A packet is sent to the MAC layer at a node when the node finishes transmitting a previous packet, or when a packet arrives while the flow is not backlogged. On arrival of packet P_i^k at the MAC layer at node *i*, the packet is stamped with start tag S_i^k, calculated as
$$S_i^k = \max\left\{v(A_i^k), F_i^{k-1}\right\}$$
where now A_i^k denotes the real time when the start tag is computed. The finish tag F_i^k, the finish tag of P_i^k is calculated as
$$F_i^k = S_i^k + \gamma \frac{L_i^k}{w_i}$$
In the above, the γ allows us to choose a suitable scale for the virtual time.

4. A node *i* wishing to transmit a packet P_i^k picks a backoff interval B_i (similar to IEEE 802.11 MAC). The chosen interval B_i is obtained as a function of S_i^k, and the current virtual time $v_i(t)$ as follows:
$$B_i = \left\lceil \eta * (S_i^k - v_i(t)) \right\rceil \quad \textit{slots}$$
where η, the *Backoff Multiplier,* is a constant. Note that because of the manner in which start tags and virtual time are computed, B_i is guaranteed to be non-negative. However, if start tag and the virtual time are identical, B_i may become equal to zero. To avoid B_i being 0, we perform one more step:

$$B_i = B_i + X, \quad X \text{ is uniformly distributed between } [1, \beta]$$

where β, the *Backoff Window*, is a positive integer. This step also reduces the probability of the backoff intervals of two nodes counting down to 0 at the same time. A variable named *Backoff Counter* is set to 0 when this step is performed.

5. If a collision occurs (because backoff intervals of two nodes count down to 0 simultaneously), then the following procedure is used to choose the new backoff window:

   - Increment *Backoff Counter* by 1.
   - Choose new B_i uniformly distributed in $\left[1,\, 2^{\mathit{Backoff\ Counter} - 1} * \beta\right]$

   The motivation in choosing small B_i after collision is as follows: The fact that this node was a winner of the contention round (although there was at least one other winner) implies that it is this node's turn to transmit next (and also the turn of the other colliding node(s)). Therefore, B_i is chosen to be small to increase the probability that a colliding node wins again soon. However, to protect against the situation when too many nodes collide, the range for B_i grows exponentially with the number of collisions.

Observe that the above protocol can be unfair for nodes whose packets collide. For instance, assume that, at the beginning of time, nodes 1, 2 and 3 pick backoff intervals of 5, 5, and 6 slots. Nodes 1 and 2 would collide when their backoff intervals count down to 0 (the interval of node 3 would count down to 1 by this time). After collision, nodes 1 and 2 pick new backoff intervals of 2 and 3 slots, respectively. In this case, node 3 would end up transmitting a packet before nodes 2 and 3, even though these two nodes should have transmitted earlier (since their original backoff intervals were smaller).

To completely eliminate such unfairness, a collision resolution protocol which guarantees colliding stations access prior to access by any other node must be used. One such protocol, which requires a slight modification to the IEEE 802.11 protocol, is as follows:

The nodes would normally wait for IFS + 1 time slots before transmitting[13]. However, a node that has experienced a collision would wait for IFS time slots and then transmit a "resolution burst" for one time slot, signaling that collision resolution is about to take place. Any node that has not experienced the collision would then back off, allowing the colliding nodes to contend for access between themselves. (An alternative is to signal the start of a collision resolution phase by transmitting a resolution burst after a smaller IFS, as compared to the IFS used by nodes that have not experienced the collision). During collision resolution, the colliding nodes may use the same algorithm as in the DFS protocol (i.e., the same procedure to update backoff interval), or some other approach. Collision resolutions would repeat until all colliding nodes have been able to access the medium. We plan to evaluate this approach in our future work.

[13] Nominally, in IEEE 802.11, nodes wait for some interval of time called an inter-frame spacing IFS before transmitting [10] .

Several other variations on the above protocol are also possible. In particular, the procedure for calculating B_i (initially and also after backoff) may be varied—these different variations would trade-off between the probability that a collision would occur (potentially resulting in unfair allocation for the colliding flows) and the overhead (incurred by the choice of large backoff intervals).

We are considering two approaches in our on-going work:

1) When initially choosing a backoff interval, we make it linearly proportional to $\textit{start tag} - \textit{virtual time}$. One possibility is to add a random component to this value—for instance, the backoff interval may be chosen to be uniformly distributed within 20% of $\textit{start tag} - \textit{virtual time}$. For instance, such an approach would benefit when the likelihood of many nodes choosing the same backoff interval is large.

2) In the above algorithm, once the backoff interval is chosen, it is decremented as in 802.11 until it reaches 0. A new backoff interval is not chosen for a packet, unless a collision occurs when its backoff interval reaches 0. This approach works since the backoff interval is chosen to be linearly proportional to $\textit{start tag} - \textit{virtual time}$.

   To see a disadvantage of this approach consider the case when there are many nodes, all with small weights. In this case, all nodes will choose large backoff intervals. With large backoff intervals, too much time spent idling when the backoff counters are counting down to 0. An alternative is to use a non-linear function to set B_i and yet emulate the relative order of backoff intervals chosen in the above scheme. For instance, a function of the form $K_1 * (1 - K_2^{(\textit{start tag} - \textit{virtual time})})$. In this case, however, after each packet transmission, the backoff counter of each pending packet will need to be recalculated, to be able to emulate the earlier scheme.

3.3.4 Performance Evaluation

We have performed a preliminary evaluation of the proposed contention-based approach. For this evaluation, $\eta = 10$ and $\beta = 4$. In the simulation environment, we set up N nodes with CBR (constant bit rate) traffic from node i to node $i+1$, where i is an even number. Thus the number of flows with N nodes (N even) is $N/2$. The choice of the flows is somewhat arbitrary, and any destination could have been chosen for each flow. Each flow is always backlogged. Packet size is 512 bytes, and $\gamma = 1/512$. We modified the *ns-2* [7] simulation package to implement the proposed contention-based protocol. The simulation object for the proposed MAC protocol was derived from that for the simulation object for the 2 Mbps IEEE 802.11 MAC.

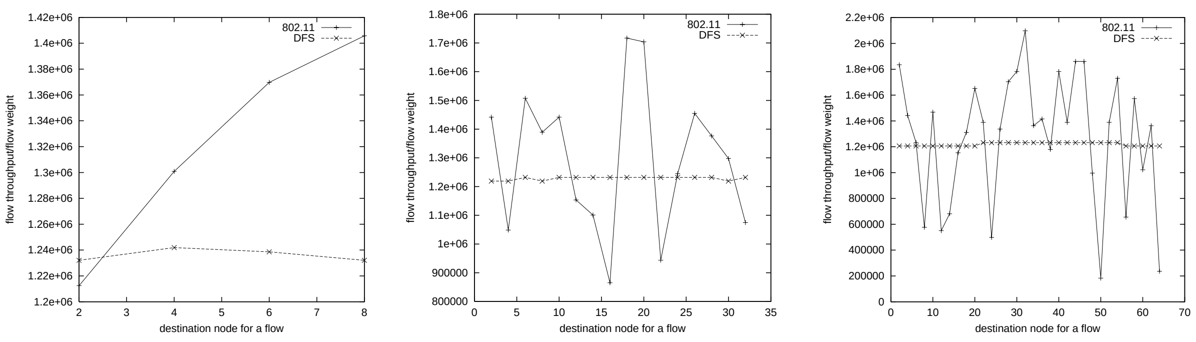
We varied the number of nodes from 2 to 128 (number of flows from 1 to 64).

Table 1 lists aggregate throughput of all flows as a function of the number of flows, for the 802.11 protocol and the proposed Distributed Fair Scheduling (DFS) protocol—for this table, the weight is identical for all flows. Specifically, weight of a flow is set to (*1/number of flows*). Performance of 802.11 is independent of the weights. The data reported here is for a single simulation (we did not average over multiple simulations, since objective of fair queuing is to provide fair allocation for each flow all the time, rather than on average).

In Table 1 observe that the aggregate throughput achieved by 802.11 is higher than the proposed protocol, since the proposed protocol tends to choose larger initial backoff intervals. However, the proposed protocol allocated the bandwidth more fairly (only when all weights are equal, comparison of the two protocols is meaningful). Figure 10 plots three graphs, for different number of flows. In each graph, two curves are plotted, one for 802.11 and one for DFS. For each curve, the numbers on the horizontal axis denote the destination node of a flow, and the corresponding value on the vertical axis plots the ratio (*throughput/weight*) for that flow—in this particular figure, the weight for each flow is (*1/number of flows*). Throughput of a flow is calculated based on the packets scheduled for that flow over a 5 second interval. Observe that the curves for DFS are close to flat, indicating that DFS is able to achieve relatively higher fairness than 802.11
.

Number of Flows	***1***	***2***	***4***	***8***	***16***	***32***	***64***
802.11	1.36	1.31	1.32	1.31	1.30	1.27	1.25
DFS	1.23	1.23	1.24	1.23	1.23	1.22	1.20

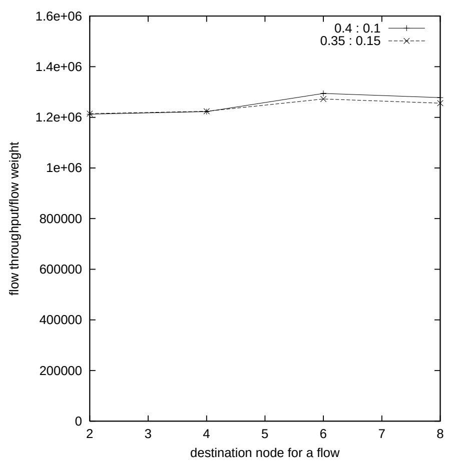
Table 1: Aggregate throughput (Mbps)



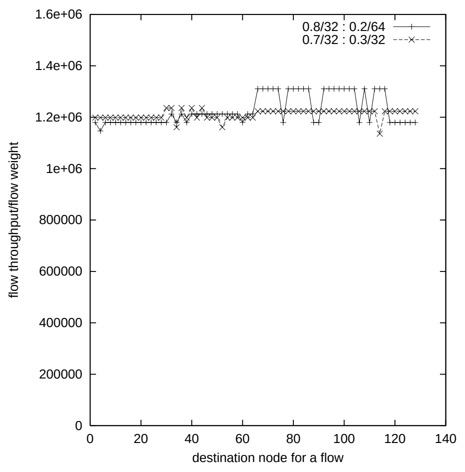
(a) 8 nodes (b) 32 nodes (c) 128 nodes

Figure 10: Comparison of 802.11 and DFS, when all weights are identical

Figure 11 presents measurements for the DFS scheme. Each graph in this figure corresponds to a different number of nodes. In each graph, each curve corresponds to a different weight assignment—weight assignment $a:b$ indicates that out of $N/2$ flows (with N nodes), the first $N/4$ were given weight a each and the remaining $N/4$ were given weight b each. Horizontal and vertical axes are similar to those in Figure 10



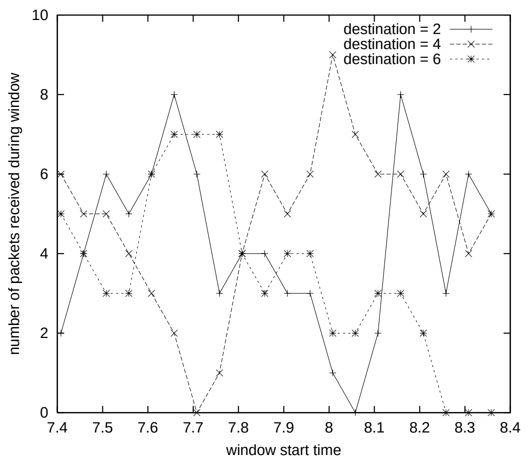
(a) 8 nodes



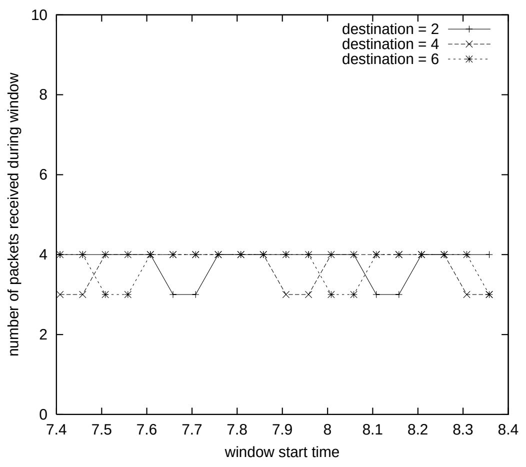
(b) 128 nodes

Figure 11: Fairness with DFS

The results reported so far evaluate long-term fairness of the proposed algorithm. To informally show that the proposed algorithm also achieves short-term fairness, we count how many packets were serviced from each flow over a window of size 0.2 second, where the window itself slides every 0.1 second. Figure 12 we compare the results for 802.11 and DFS—the four graphs correspond to 3 flows out of 8 for the case of 16 nodes. For DFS, the weight of all flows is identical.



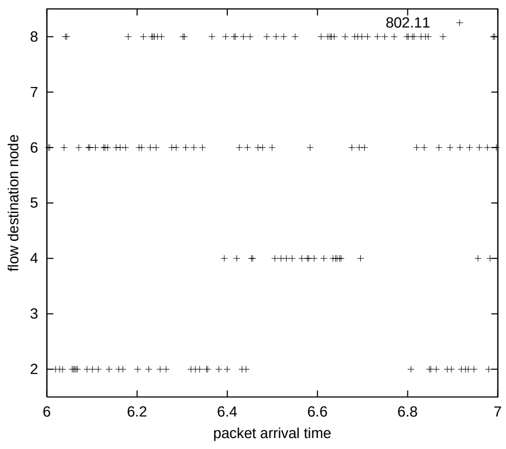
(a) 802.11



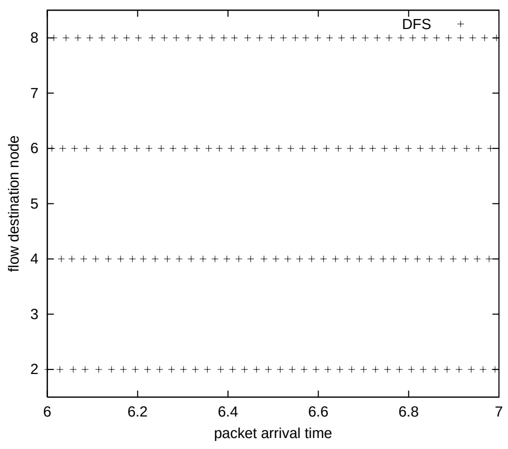
(b) DFS

Figure 12: Packet delivery times

Observe that, with 802.11, the counts have a lot more variation than with DFS. Figure 13 plots delivery times for packets for the case of 16 nodes (8 flows). Observe that the packet arrival times with DFS are significantly less bursty (or closer to uniform distribution) than those for 802.11.



(a) 802.11



(b) DFS

Figure 13: Packet delivery time

3.4 Dynamic Adaptation of Weights

In the above discussion, we assumed that the weight of each flow is constant, and predefined. It is conceivable that in some environments it may be desirable to dynamically determine a suitable weight for each flow. For instance, the weight of a flow may be chosen to be proportional to the recent demand of that flow (i.e., recent arrival rate of data on the flow)[14]. The *DFS protocol* described above is flexible in that it allows a different weight for each packet without increasing protocol complexity.

3.4.1 Method 1: Adapting Weights based on Input Rate

The first method estimates average rate of arrival of data on a given flow to determine its weight. Any estimator of arrival rate may be used. We consider the following method. When a packet of size L arrives on a given flow, the rate of arrival $r(\tau)$ at time τ for that flow is updated as:

$$r(\tau) = \frac{t\,r(t) + \xi\ L}{\tau}$$

In the above expression, ξ is a constant that determines sensitivity of the rate estimate to short-term changes in arrival pattern, and $r(t)$ is the rate as determined at some previous packet arrival time instant t.

Given an estimate of the rate of arrival, the weight is determined as:

$$w(\tau) = \frac{r(\tau)}{Normalizing\ \ factor}$$

[14] Computing weights of each flow is less desirable for the case of single node, multiple flows inside a host, due to the processing overhead, and for the case of centralized PFQ algorithms for LANs due to the undesirable additional communications overhead. However, for the case of DFS, specially for the case of single flow per host model this overhead is generally manageable.

where $Normalizing \ \ factor$ is a constant. For instance, in the case of a 2 Mbps IEEE 802.11 LAN, the maximum achievable throughput is about 1.3 Mbps (with packet size 512 bytes). Thus, we may use a normalizing factor of 1.3 Mbps. With such a normalizing factor, the weight approximately represents the arrival rate as a fraction (or multiple) of maximum achievable throughput.

One disadvantage of the above procedure for updating the arrival rate is that if a connection stays idle for a very long duration of time before becoming active again, it may take some time for the estimated rate to converge on the actual rate. To avoid such a situation, a lower bound may be imposed on the estimated rate (alternatively, an upper bound may be imposed on the value of time since last measurement used in estimating $r(\tau)$).

3.4.2 Method 2: Adapting Weights based on Average Queue Size

An alternative method is to adapt the weight of a flow such that it is proportional to the number of pending packets for that flow. First, the number of packets in the queue for the flow must be estimated -- many different approaches to estimating the average queue size are possible including the low pass filter proposed for RED [31] or the slightly modified version of it proposed in FRED [32]. Having estimated the average queue size, the weight for the flow may be defined as:

$$w(\tau) = \frac{Avg \quad queue \quad size(\tau)}{Max \quad allowed \quad queue \quad size}$$

4 Summary

This report discussed several issues related to fair scheduling in broadcast networks. In particular, difficulties in defining fairness in multi-hop wireless networks are identified, a *Distributed Fair Scheduling* algorithm to achieve fairness on local area networks is evaluated and methods for dynamically adapting weights of flows on each node are proposed.

References

[1] G. Anastasi, L. Lenzini and E. Mingozzi, "Stability and Performance Analysis of HIPERLAN," *Proceedings of INFOCOM '88,* pp. 134-141 (1998)

[2] J. C. R. Bennett and H. Zhang, "WF^{2}Q: Worst-case Fair Weighted Fair Queueing," *Proceedings of INFOCOM '96,* pp. 120-128 (March 1996)

[3] J. C. R. Bennett and Hui Zhang, "Hierarchical Packet Fair Queueing Algorithms," *Proceedings of SIGCOMM '96,* pp. 43-56 (August 1996)

[4] V. Bharghavan, S. Lu and T. Nandagopal, "Fair Queueing in Wireless Networks: Issues and Approaches," *IEEE Personal Communications Magazine* (February 1999)

[5] G. L. Choudhury and S. S. Rappaport, "Priority Access Schemes Using CSMA/CD," *IEEE Transactions on Communications,* vol. COM-33, no. 7, pp. 620-626 (July 1985)

[6] A. Demers, S. Keshav and S. Shenkar, "Analysis and Simulation of a Fair Queueing Algorithms," *Proceedings of SIGCOMM '89*, pp. 3-12 (August 1989)

[7] K. Fall and K. Varadhan, "*ns* — Notes and Documentation," *VINT Project*, University of California @ Berkeley and LBNL (1997)

[8] R. Garces and J. J. Garcia-Luna-Aceves, "Collision Avoidance and Resolution Multiple Access (CARMA)," *Journal on Cluster Computing* (1998)

[9] P. Goyal, H. M. Vin and H. Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 690-704 (October 1997)

[10] IEEE, "IEEE Std 802.11 – Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," (1997)

[11] A. Kamerman and L. Monteban, "WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed Band," *Bell Labs Technical Journal*, pp. 118-133 (Summer 1997)

[12] S. Keshav, "On the Efficient Implementation of Fair Queueing," *Journal of Internetworking: Research and Experience*, vol. 2, no. 3, pp. 57-73 (September 1991)

[13] S. Keshav, "An Engineering Approach to Computer Networking," *Addison Wesley*, 1997

[14] C. C. Ko, K. M. Lye and W. C. Wang, "Simple Priority Scheme for Multichannel CSMA/CD Local Area Networks," *IEE Proceedings*, vol. 137, no. 6, pp. 365-370 (December 1990)

[15] Y. Ko and N. H. Vaidya, "Medium Access Control Protocols using Directional Antennas in Ad Hoc Networks," Computer Science, Texas A&M University, 99-010 (May 1999)

[16] C. R. Lin and M. Gerla, "Asynchronous Multimedia Multihop Wireless Networks," *Proceedings of INFOCOM '97* (1997)

[17] S. Lu, V. Bharghavan and R. Srikant, "Fair Scheduling in Wireless Packet Networks," *Proceedings of SIGCOMM '97*, pp. 63-74 (September 1997)

[18] S. Lu, T. Nandagopal and V. Bharghavan, "A Wireless Fair Service Algorithm for Packet Cellular Networks," *Proceedings of MobiCom '98*, pp. 10-20 (October 1998)

[19] S. Lu, T. Nandagopal and V. Bharghavan, "Design and Analysis of an Algorithm for Fair Service in Error-Prone Wireless Channels," *Journal of Wireless Networks* (February 1999)

[20] T. Nandagopal, S. Lu and V. Bharghavan, "A Unified Architecture for the Design and Evaluation of Wireless Fair Queueing Algorithms," *Proceedings of MobiCom '99*, pp. 132-142 (August 1999)

[21] T. S. Ng, I. Stoica and H. Zhang, "Packet Fair Queueing: Algorithms for Wireless Networks with Location-Dependent Errors," *Proceedings of INFOCOM '98* (March 1998)

[22] A. K. Parekh and R. G. Gallagher, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-node Case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344-357 (June 1993)

[23] A. K. Parekh and R. G. Gallagher,"A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple-node Case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 2, pp. 137-150 (March 1994)

[24] P. Ramanathan and P. Agrawal, "Adapting Packet Fair Queueing Algorithms to Wireless Networks," *Proceedings of MobiCom '98*, pp. 1-9 (October 1998)

[25] S. M. Sharrock and D. H. Du, "Effcient CSMA/CD - Based Protocols for Multiple Priority Classes," *IEEE Transactions on Computers*, vol. 31, no. 7, pp. 943-954 (July 1989)

[26] M. Shreedhar and G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," *Proceedings of SIGCOMM'95* (August 1995)

[27] J. L. Sobrinho and A. S. Krishnakumar, "Real-Time Traffic over the IEEE 802.11 Medium Access Control Layer," *Bell Labs Technical Journal*, pp. 172-187 (Autumn 1996)

[28] J. L. Sobrinho and A. S. Krishnakumar, "Quality-of-Service in Ad Hoc Carrier Sense Multiple Access Networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1353-1368 (August 1999)

[29] Y. Sun and K. Chen, "A Multi-Layer Collision Resolution Multiple Access Protocol for Wireless Networks," *Jounral on Wireless Networks*, pp. 353-364 (1998)

[30] K. J. Negus, J. Waters, J. Tourrilhes, C. Romans, J. Lansford, and S. Hui, "HomeRF and SWAP: Wireless Networking for the Connected Home," *Mobile Computing and Communications Review*, vol. 2, no. 4, pp. 28-37 (October 1998)

[31] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-413 (August 1993)

[32] D. Lin and R. Morris, "Dynamics of Random Early Detection," *Proceedings of SIGCOMM '97*, pp. 127-137 (September 1997)

[33] I. Chlamtac and A. Lerner, "Fair Algorithms for Maximal Link Activation in Multihop Radio Networks, " IEEE Transactions on Communications, vol. COM-35, no. 7, pp. 739-746 (July 1987).