# Distributed Fair Scheduling in a Wireless LAN

Nitin H. Vaidya
Texas A&M University
vaidya@cs.tamu.edu

Paramvir Bahl
Microsoft Research
bahl@microsoft.com

Seema Gupta
Texas A&M University
seemag@cs.tamu.edu

## ABSTRACT

Fairness is an important issue when accessing a shared wireless channel. With *fair scheduling*, it is possible to allocate bandwidth in proportion to *weights* of the packet *flows* sharing the channel. This paper presents a fully *distributed* algorithm for fair scheduling in a wireless LAN. The algorithm can be implemented without using a centralized coordinator to arbitrate medium access. The proposed protocol is derived from the Distributed Coordination Function in the IEEE 802.11 standard. Simulation results show that the proposed algorithm is able to schedule transmissions such that the bandwidth allocated to different flows is proportional to their weights. An attractive feature of the proposed approach is that it can be implemented with simple modifications to the IEEE 802.11 standard.

## 1. INTRODUCTION

Wireless communication technology has gained widespread acceptance in recent years. Wireless local area networks have come into greater use, with the advent of the IEEE 802.11 standard [12]. Fairness is an important issue when accessing a shared wireless channel. With *fair scheduling*, different flows sharing a wireless channel can be allocated bandwidth in proportion of their "weights". This paper presents a *distributed* medium access control (MAC) protocol for fair scheduling in a wireless LAN. Although IEEE 802.11 wireless MAC [12] is not *fair* (particularly on short time-scales), the proposed protocol is derived from the Distributed Coordination Function (DCF) in IEEE 802.11. An attractive feature of our approach is that it can be implemented with simple modifications to IEEE 802.11. This section discusses the motivation for considering *distributed* protocols, and elaborates on the definition of *fairness.*

### 1.1 Centralized and Distributed Protocols

Wireless transmissions by hosts within proximity of each other can interfere. Therefore, several medium access control (MAC) protocols for wireless networks have been proposed in the past. In general, MAC protocols may be divided into two types:

- Centralized: In centralized protocols, a designated host (often referred to as base station or access point) coordinates access to the wireless medium. A node wanting to transmit must wait until permission to transmit is granted by the coordinator node – the mechanisms for requesting and granting such permission may differ in different protocols. Point Coordination Function (PCF) in IEEE 802.11 is an example of the centralized approach.
- Distributed: In distributed protocols, a coordinator is not needed to arbitrate access to the wireless medium. For instance, in the CSMA (carrier sense multiple access) protocol, a node wishing to transmit a packet does so only if it does not hear another on-going transmission. CSMA protocol is fully distributed, since each node independently determines whether to transmit a packet or not. Distributed Coordination Function (DCF) in IEEE 802.11 is an example of the distributed approach.

There are several benefits of using a distributed approach as compared to a centralized approach:

- In the centralized approach, if a node cannot communicate with the coordinator, then it cannot transmit any packets. On the other hand, with a distributed protocol, if a node cannot communicate with some nodes, it may still be able transmit packets to other nodes.
- In the centralized approach, the coordinator has the responsibility of keeping track of the *state* information for nodes on the LAN. In distributed protocols, this overhead can be eliminated.
- In a centralized approach, it is difficult to use a battery-powered node as the coordinator, since the coordinator will fail if the battery runs out. With failure-prone coordinators, other nodes must be able to reliably detect failure of the coordinator, and elect a new coordinator.

Keeping the above issues in mind, this paper develops a distributed approach for fair scheduling.

## 1.2 Fair Queueing and Other Related Work

Much research has been performed on "fair queueing" algorithms for achieving a *fair* allocation of bandwidth on a shared link [1, 4, 10, 14, 19, 22]. Consider the system shown in Figure 1, where a node maintains several *queues* (or *flows*) which store packets to be transmitted on an output link. A fair queueing algorithm is used to determine which flow to serve next, so as to satisfy a certain fairness criterion. By design, these fair queuing algorithms are centralized, since they are executed on a single node (for instance, a switch or router) which has access to all information about the flows.

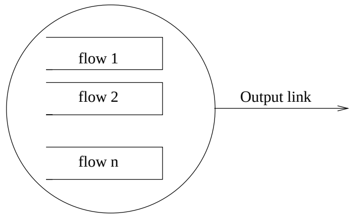



**Figure 1: A node with several flows sharing a link**

Fair queueing algorithms in literature typically attempt to approximate the Generalized Processor Sharing (GPS) discipline [19]. When using the GPS discipline, a server serves, say, $n$ flows each characterized by a positive *weight*; let $\phi_i$ denote the weight associated with flow $i$ ($i = 1, \cdots, n$). Let $W_i(t_1, t_2)$ be the amount of flow $i$ traffic served in the interval $[t_1, t_2]$. Then, for a GPS server [19], if flow $i$ is backlogged[1] throughout $[t_1, t_2]$, the following condition holds:

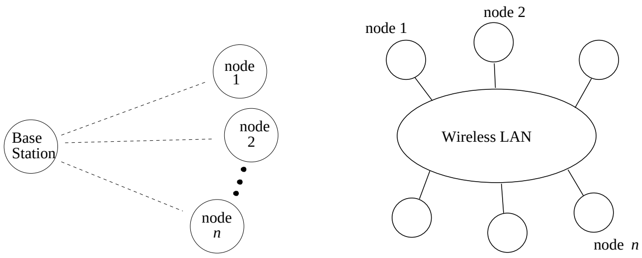
$$\frac{W_i(t_1,t_2)}{W_j(t_1,t_2)} \geq \frac{\phi_i}{\phi_j}, \quad \forall j \tag{1}$$

Equality holds above if flow $j$ is also backlogged in interval $[t_1, t_2]$. Note that the above condition is valid regardless of how small the interval $[t_1, t_2]$ is. This implies that the GPS server can "interleave" data from different flows with an *arbitrarily fine* granularity. The GPS discipline cannot be accurately implemented in practice, since data transmitted on real networks is *packetized*. This observation led to development of several *packet* fair queuing algorithms which approximate GPS under the constraint that each packet must be transmitted as a whole [1, 4, 10, 14, 19, 22]. These protocols are centralized by design, as noted above.

There has also been some work on achieving fairness using distributed MAC protocols for wireless networks [2, 8]. However, past work on incorporating fairness into distributed protocols has been limited in that these protocols attempt to provide *equal* share of bandwidth to different nodes (essentially, node weights are implicitly assumed to be equal). There has been work on distributed protocols that take *priorities* into account when performing medium access control [3, 21]. However, these protocols do not perform *fair* allocation of bandwidth. Interesting work on a distributed scheduling algorithm for real-time traffic on a wireless LAN has also been performed [23]. This work, however, assumes that a flow transmits packets with a constant rate. Such assumptions cannot be made when performing fair scheduling.

**Location-Dependent Errors:** In recent years, researchers have also considered the use of fair queueing in the wireless cellular environment illustrated in Figure 2(a). Although existing centralized algorithms may be applied to the wireless environment (with the base station acting as the coordinator), it has been observed that fairness achieved by these algorithms may suffer in presence of *location-dependent* errors [18] – with *location-dependent* errors, while error-free transmission may be possible between a given host and the base station, transmissions between another host and the base station may be corrupted by errors. In this case, some mechanism to "compensate" hosts whose packets are corrupted by errors should be incorporated. Many approaches for improving fairness in presence of location-dependent errors have been developed [16, 17, 18, 20]. These approaches are *centralized* and require the base station to coordinate access to the wireless channel.

[1]A queue (or flow) is said to be backlogged if it is not empty.




(a) In centralized approaches, base station coordinates medium access

(b) In distributed approaches, all nodes have identical responsibilities

**Figure 2: Wireless environments**

# 2. PRELIMINARIES

Objective behind this work was to develop a fair scheduling MAC protocol for a wireless LAN (illustrated in Figure 2(b)), with the following properties:

- The protocol must be *fully distributed* in that no single node should have any special responsibility.
- Each node should be able to independently determine when to transmit a packet, *without* knowing the state of (or existence of) flows at other nodes – state of a flow includes information such as weight of the flow, whether the flow is backlogged or not, and time of arrival of packets on the flow.
- Maintain compatibility or close resemblance to an existing wireless MAC standard, to make it easier to implement the proposed protocol.

## 2.1 Proposed Approach

We observe that many centralized fair queueing algorithms behave as follows:

- The coordinator maintains a "virtual clock" – different algorithms differ in how the virtual clock is updated.

- *Start* and *finish* tags are assigned to each packet arriving on each flow. Packets are scheduled for transmission in the order of either finish tags [9, 4, 1] or start tags [10].

Two significantly different approaches are used for updating the virtual clock:

- In one approach, the *rate* of increase of virtual time is a function of the set of backlogged flows [4, 1].
- In the alternative approaches, the virtual clock is updated to be equal to either start tag [10] or finish tag [9] of the most recent packet in service.

The first approach above (for updating virtual clocks) potentially allows the fair queuing algorithms to match GPS more closely [1]. However, in the second approach, the virtual clock can be updated without knowing which flows are backlogged [9, 10]. Due to this property, the second approach is more suitable for a distributed implementation. Next two subsections describe a *centralized* fair queueing algorithm, and the IEEE 802.11 MAC protocol, which together form the basis for the proposed fair scheduling protocol.

## 2.2 Self-Clocked Fair Queueing (SCFQ)

The algorithm proposed here was designed in an attempt to emulate Self-Clocked Fair Queueing (SCFQ) [9] in a distributed manner. Two important issues are worth noting:

- The proposed technique to implement distributed fair scheduling can also be extended to other fair queueing algorithms, such as Start-Time Fair Queueing (SFQ) [10].
- Although our intention was to emulate SCFQ, the distributed implementation behaves somewhat differently, as discussed later in Sections 4.1 and 4.4.

Now we briefly describe the centralized SCFQ algorithm [9] which assumes the architecture shown in Figure 1. A virtual clock is maintained by the central coordinator, and $v(t)$ denotes the *virtual* time at real 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. Let $F_i^0 = 0$, $\forall i$.

1. On arrival of packet $P_i^k$, the packet is stamped with start tag $S_i^k$, calculated as
$$S_i^k = maximum\{v(A_i^k), F_i^{k-1}\}$$
Also, $F_i^k$, the finish tag of $P_i^k$ is calculated as
$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}$$
2. Initially, the virtual clock is set to 0, i.e., $v(0) = 0$. The virtual time is updated only when a new packet is transmitted. When a packet begins transmission on the output link, the virtual clock is set equal to the finish tag of that packet.
3. Packets are transmitted on the link in the increasing order of their finish tags. Ties are broken arbitrarily.

As noted in Step 1 above, in the SCFQ algorithm (and, also in other algorithms, such as SFQ [10], WFQ [4], WF2Q [1], etc.), the start and finish tags are calculated when a packet *arrives* in a flow. An alternative approach is to calculate the start tag when a packet reaches the *front* of its flow – that is, for a packet $P_i^k$ in flow $i$, start and finish tags are calculated only after all packets that arrived in flow $i$ before packet $P_i^k$ have been serviced. If this approach were to be used, then calculation of the start tag above should be modified as follows:

- Let $f_i^k$ denote the real time when packet $P_i^k$ reaches the *front* of its flow. If $P_i^k$ arrives on an empty flow, then $f_i^k = A_i^k$; else $f_i^k$ will denote the real time when $P_i^{k-1}$ finishes service. On arrival of packet $P_i^k$ at the front of its flow, the packet is stamped with start tag $S_i^k$, calculated as
$$S_i^k = v(f_i^k) \tag{2}$$

The finish tag is calculated as before, as $F_i^k = S_i^k + L_i^k/\phi_i$. It is a simple exercise to verify that, for the SCFQ algorithm, this new procedure and the earlier procedure result in the same start and finish tags for all packets. In our distributed implementation, however, we emulate the latter procedure.

## 2.3 IEEE 802.11 MAC : Distributed Coordination Function

The MAC specified in IEEE 802.11 standard cannot perform fair allocation, particularly on short time scales (even if we assume that all flows have equal weights). However, using a mechanism similar to the Distributed Coordination Function (DCF) in IEEE 802.11, the proposed protocol is able to achieve significantly better fairness.

We now briefly present salient features of the Distributed Coordination Function (DCF) in IEEE 802.11. CSMA/CA (collision avoidance) mechanism is incorporated in DCF – a similar mechanism is also used in the proposed protocol. When a node $i$ wishes to transmit a packet, it chooses a "backoff" interval equal to $B_i$ *slots*.[2] Specifically, $B_i$ is chosen uniformly distributed in the interval $[0, cw]$, where $cw$ is size of the so-called *contention window*. $cw$ at node $i$ is reset to a value $CW_{min}$ at the beginning of time, and 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. Then, while the medium is idle, $B_i$ is decremented by 1 after each slot time.[3] 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 for the intended

[2] A slot is a fixed interval of time defined in IEEE 802.11.
[3] Actually, node $i$ waits for an interval known as an inter-frame spacing [12], before starting to decrement $B_i$. We will omit such details in this discussion. However, our simulation model implements these details accurately.

destination of the packet. The destination node, on receiving the RTS, sends a Clear-to-Send (CTS) packet. The node $i$, on receipt of the CTS packet, transmits the data packet. The receiver node, on receipt of data, sends an acknowledgement (ACK). 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. 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, node $i$ doubles its contention window size $cw$, picks a new $B_i$ uniformly distributed over $[0, cw]$, and repeats the above procedure.

## 3. PROPOSED DISTRIBUTED FAIR SCHEDULING (DFS) PROTOCOL

The proposed *Distributed Fair Scheduling* (DFS) protocol is based on the IEEE 802.11 MAC and SCFQ:

- The DFS protocol borrows on SCFQ's idea of transmitting the packet whose finish tag is smallest, as well as SCFQ's mechanism for updating the virtual time.
- A distributed approach for determining the smallest finish tag is employed, using the *backoff interval* mechanism from IEEE 802.11 MAC. The essential idea is to choose a backoff interval that is proportional to the finish tag of packet to be transmitted. Several implementations of this idea are possible, as discussed below.

We now describe the proposed approach. In our discussion and simulations, we assume that all packets at a node belong to a single flow – the proposed algorithm can be easily extended when multiple queues are maintained at each node (as discussed later in Section 4.2).

Each node $i$ maintains a local virtual clock, $v_i(t)$, where $v_i(0) = 0$. Now, $P_i^k$ represents the $k$-th packet arriving at the flow at node $i$ on the LAN.

- Each transmitted packet is tagged with its finish tag.
- When at time $t$ node $i$ hears or transmits a packet with finish tag $Z$, node $i$ sets its virtual clock $v_i$ equal[4] to $maximum(v_i(t), Z)$.
- Start and finish tags for a packet are **not** calculated when the packet arrives. Instead, the tags for a packet are calculated when the packet reaches the front of its flow. When packet $P_i^k$ reaches the *front* of its flow at node $i$, the packet is stamped with start tag $S_i^k$, calculated as (similar to Equation 2 for the SCFQ algorithm), $S_i^k = v(f_i^k)$, where $f_i^k$ denotes the real time when packet $P_i^k$ reaches the front of the flow.

  Finish tag $F_i^k$ is calculated as follows, where appropriate choice of the *Scaling_Factor* allows us to choose a suitable scale for the virtual time.

$$\begin{aligned} F_i^k &= S_i^k + Scaling\_Factor * \frac{L_i^k}{\phi_i} \\ &= v(f_i^k) + Scaling\_Factor * \frac{L_i^k}{\phi_i} \end{aligned}$$

- The objective of the next step is to choose a backoff interval such that a packet with smaller finish tag will ideally be assigned a smaller backoff interval. This step is performed at time $f_i^k$. Specifically, node $i$ picks a backoff interval $B_i$ for packet $P_i^k$, as a function of $F_i^k$ and the current virtual time $v_i(f_i^k)$, as follows:

$$B_i = \left\lfloor F_i^k - v(f_i^k) \right\rfloor \text{ slots,} \tag{3}$$

  Now, observe that, since $F_i^k = v(f_i^k) + Scaling\_Factor * \frac{L_i^k}{\phi_i}$, the above expression reduces to:

$$B_i = \left\lfloor Scaling\_Factor * \frac{L_i^k}{\phi_i} \right\rfloor \tag{4}$$

  Finally, to reduce the possibility of collisions, we randomize the $B_i$ value chosen above as follows:

$$B_i = \lfloor \rho * B_i \rfloor \tag{5}$$

  where $\rho$ is a random variable with mean 1 – in our simulations, $\rho$ is uniformly distributed in $[0.9, 1.1]$.

  When this step is performed, a variable named *CollisionCounter* is reset to 0.

- Collision handling: If a collision occurs (because backoff intervals of two or more nodes count down to 0 simultaneously), then the following procedure is used. Let node $i$ be one of the nodes whose transmission has collided with some other node(s). Node $i$ chooses a new backoff interval as follows:
  - Increment *CollisionCounter* by 1.
  - Choose new $B_i$ uniformly distributed in $\left[1, 2^{CollisionCounter-1} * CollisionWindow\right]$, where *CollisionWindow* is a constant parameter.

  If *CollisionWindow* is chosen to be small, the above procedure tends to choose a relatively small $B_i$ (in the range $[1, CollisionWindow]$) after the first collision for a packet. The motivation for choosing small $B_i$ after the first collision is as follows: The fact that node $i$ was "a potential winner" of the contention for channel access indicates that it is node $i$'s turn to transmit in the near future. Therefore, $B_i$ is chosen to be small to increase the probability that node $i$ 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 consecutive collisions.

The above protocol has two potential shortcomings:

- The DFS protocol can exhibit short-term unfairness for some nodes when their packets collide. For instance, assume that, at the beginning of time, nodes

[4]The virtual clock update mechanism in DFS differs somewhat from that in SCFQ. Due to potential collision between packets in the distributed implementation, occasionally a packet with a smaller finish tag may be transmitted before a packet with a greater finish tag. To ensure that virtual clocks are non-decreasing, $max(v_i(t), Z)$ is used in this step. Incidentally, as discussed later in Section 4.1, DFS can be implemented *without* maintaining virtual clocks at the nodes.

1, 2 and 3 pick backoff intervals of 25, 25, and 26 slots, respectively. Nodes 1 and 2 would collide when their backoff intervals count down to 0 (the backoff interval of node 3 would count down to 1 slot by this time). After collision, nodes 1 and 2 pick new backoff intervals of, say, 2 and 3 slots, respectively. In this case, node 3 would end up transmitting a packet before nodes 1 and 2, even though these two nodes should have transmitted earlier (since their original backoff intervals were smaller).

To eliminate such unfairness, a collision resolution protocol which *guarantees* colliding stations access prior to access by any other node (or, a protocol which ensures this with a high probability) must be used. Protocols for collision resolution have been proposed in the past [7]. Analogous approaches may be used in conjunction with our algorithm as well. Guaranteeing "near-perfect" collision resolution, however, may add to the overhead – therefore, for our performance evaluation, we consider the DFS algorithm presented above, without using such a mechanism.

- Observe that in DFS, duration of the backoff interval is inversely proportional to weight of a flow. When the weights of *backlogged* flows are small (weights of presently idle flows may be large), the duration of the backoff intervals can become large. This leads to long durations of idle time, when the nodes are counting down the backoff intervals to 0. To address this problem, we now present an *exponential mapping* scheme to translate finish tags into backoff intervals.

## 3.1 Exponential Mapping Scheme

We will refer to the scheme presented above for calculating the backoff interval as the *linear* scheme (or linear mapping). From Equations 3 and 4, observe that in the *linear* scheme, backoff interval $B_i$ is a linear function of finish tag, and directly proportional to (1/flow weight). This can make the backoff intervals large, when flow weights are small, as noted above. We consider an alternative approach to obtain the backoff interval, as follows (other alternatives are also possible).

Let $\Delta$ denote the backoff interval obtained in Equation 5 using the linear scheme described above. When using the exponential scheme, we apply another function $\gamma(\Delta)$ to obtain the actual backoff interval $B_i$ to be used for medium access. Function $\gamma(\Delta)$ is defined in Figure 3. In the definition of $\gamma(\Delta)$ in Figure 3, note that, *Threshold*, $K_1$ and $K_2$ are constant parameters. Use of the $\gamma$ function has the impact of compressing large $\Delta$ values into a smaller range – this has an advantage and a disadvantage:

- The advantage is that the time spent in counting down backoff intervals is reduced, potentially improving performance when weights of backlogged flows are small.

  EXAMPLE 1. *Consider an example of two flows with weights 0.01 and 0.02, respectively – it may be the case that there are several other flows, however, let us assume that the other flows are not backlogged presently. With the* linear *approach, backoff intervals would be inversely proportional to the weights. With packets of size 1000 bytes, and* Scaling_Factor = *1/100, the linear approach may yield backoff intervals of 1000 slots and 500 slots, respectively, for the two flows. Now, for the exponential scheme, suppose* Threshold = 80, $K_1$ = *80, and* $K_2 = 0.002$. *Then, the corresponding exponentially mapped backoff interval would be* $\gamma(1000) = 147$ *and* $\gamma(500) = 125$ *slots, respectively. Thus, the backoff interval of flow 2 would count down to 0 much sooner with the exponential mapping, as compared to the linear mapping.* □

- The disadvantage is that, since a larger range of *linear* backoff intervals is "compressed" into a smaller *exponential* range, the likelihood of collisions can increase with the exponential scheme. For instance, $\gamma(990) = \gamma(1000) = 147$ – therefore, two nodes which simultaneously begin counting down from initial backoff intervals of 990 and 1000 slots when using the *linear* scheme, would instead both start counting down from 147 slots when using the *exponential* scheme. If the linear scheme were to be used, these two nodes would not collide, however, with the exponential mapping scheme they would collide.

  To reduce the possibility for such additional collisions, when defining $\gamma$ we introduced *Threshold* as a lower bound (on backoff interval) below which the exponential function is not applied – thus, the final value of $B_i$ may belong to the *linear range* (between 1 and *Threshold*) or the *exponential range* (above *Threshold*).

  A small *Threshold* would result in poorer fairness but higher throughput, while a larger *Threshold* would yield better fairness but poorer throughput. Thus, by choosing the appropriate *Threshold*, a trade-off between fairness and throughput can be obtained.

The above exponential mapping scheme needs to be augmented to incorporate a *recalculation* procedure, as discussed below.

**Recalculation of Backoff Intervals:** Unlike the case of linear mapping, additional care needs to be taken to ensure fair allocation in case of exponential mapping – in particular, the backoff intervals must be "recalculated" after each packet transmission to maintain fairness. Let us explain this using the following example.

EXAMPLE 2. *Consider two flows, flow 1 at node 1, and flow 2 at node 2, with weights 1.0 and 0.05, respectively. Assume that both flows begin with several queued packets of identical size at time 0. Let the packet size be 1000 bytes, and the* Scaling_Factor *be 0.01. Then,* Scaling_Factor∗ *packetsize*/flow weight *will be 10 slots for flow 1, and 200 slots for flow 2. For simplicity, let us assume that the random multiplier (i.e.,* $\rho$*) used for all packets is 1.0 in this example. Therefore, flow 1 will pick a backoff interval of 10 slots for all its packets, and flow 2 will pick a backoff interval of 200 slots for all its packets. As a result, on average, flow 2 will transmit one packet for every 20 packets transmitted by flow 1 – this is consistent with the assigned weights. Now, if the exponential scheme were to be used with* Threshold = 80

$$B_i \;=\; \gamma(\Delta) = \begin{cases} \Delta, & \text{if } \Delta < \mathit{Threshold} \\ \left\lfloor \mathit{Threshold} + K_1 * \left(1 - e^{-K_2 * (\Delta - \mathit{Threshold})}\right) \right\rfloor, & \mathit{otherwise} \end{cases} \qquad (6)$$

**Figure 3: Function $\gamma$ : *Threshold*, $K_1$ and $K_2$ are constant parameters. In our simulations, we use $K_1$ = *Threshold*.**

*slots, $K_1 = 1000$ and $K_2 = 0.002$, then, flow 1 will continue to use backoff interval of 10 slots, but flow 2 will pick a backoff interval of $\gamma(200)$=97. Now, unless some precaution is taken, flow 2 will transmit a packet after approximately 9 or 10 packets transmitted by flow 1, on average – this is* inconsistent *with the assigned weights.* □

The above example illustrates that, unless modified, the exponential mapping scheme presented above can result in unfair bandwidth allocation. To avoid such unfairness, the backoff intervals in the *exponential range* must be *recalculated* after each packet transmission on the wireless channel. We now describe our recalculation procedure (other alternatives for recalculation are also possible, but not discussed here for brevity). When using the recalculation procedure, the $\Delta$ value for a given pending packet may be recalculated many times.

Consider a packet $P$ that is being transmitted on the channel presently – let the most recent value of $\Delta$ for this packet be $\Delta_{current}$. Then, to allow recalculations to be performed, when packet $P$ is transmitted, we tag it with the value $\Delta_{current}$. For instance, in Example 2 above, node 1 might tag its transmitted packet with $\Delta_{current} = 10$. Now, when some node $i$ hears a packet transmitted by node $j$, node $i$ updates the $\Delta$ and $B_i$ for its pending packet (if any) as shown in Figure 4. (Please note that due to vagaries of our text formatting software, the figures in this paper may not appear in order of their sequence numbers.)

The final step in Figure 4 recalculates backoff interval. Node $i$ then begins to count down from this new value of $B_i$.

In Example 2, flows 1 and 2 initially set $\Delta$ to 10 and 200 slots, respectively, and the backoff intervals to 10 and 97 slots, respectively, as discussed above. Now, when flow 1 transmits its packet after counting down the backoff interval from 10 to 0, it tags the transmitted packet with $\Delta_{current} = 10$. On hearing this packet, node 2 updates its $\Delta$ as 200-10=190, and *recalculates* the backoff interval as $\gamma(190)$. (Now, for the packet on flow 2, $\Delta_{current} = 190$.)

In the above example, since the backoff interval of flow 1 was in the linear range, for its transmitted packets, most recently calculated values of $\Delta$ and the chosen backoff interval are equal – however, in general, this may not be the case. For instance, if only flow 2 was backlogged in the above example (i.e., flow 1 does not attempt to transmit), then flow 2 will start with backoff interval of 97 slots and $\Delta$ = 200 slots, and eventually transmit a packet – this packet would then have been tagged with its $\Delta_{current} = 200$.

## 3.2 Other Mappings

In general, any increasing function can be used to map $\Delta$ values to backoff intervals, similar to the $\gamma$ exponential mapping function defined earlier. Note that although the linear and exponential mapping functions are increasing, they are not strictly *monotonically* increasing functions, due to the fact that backoff intervals must be integers. This can result in many $\Delta$ values being mapped to the same backoff interval – the frequency of such occurrences depends on how much "compression" is performed by the mapping function. Observe that the exponential function results in a significantly greater compression than the linear mapping. As a compromise between these two possibilities, in our evaluation, we also consider another mapping $\Psi(\Delta)$ defined in Figure 5. We will refer to the mapping in Figure 5 as the *square-root*

$$B_i = \Psi(\Delta) = \begin{cases} \Delta, & \text{if } \Delta < \mathit{Threshold} \\ \left\lfloor \sqrt{\mathit{Threshold} * \Delta} \right\rfloor, & \mathit{otherwise} \end{cases}$$

**Figure 5: Function $\Psi$**

*mapping*. Procedure for using the square-root mapping is identical to that for exponential mapping, except that $\Psi(\Delta)$ is used instead of $\gamma(\Delta)$. The *recalculation* procedure is also similar to that for the exponential mapping, with the only difference being that $\Psi(\Delta)$ is used instead of $\gamma(\Delta)$.

Figure 6 illustrates the three mappings considered in this paper. Clearly, many other alternatives for the mapping are also possible. In this paper, however, only the above mappings are evaluated.

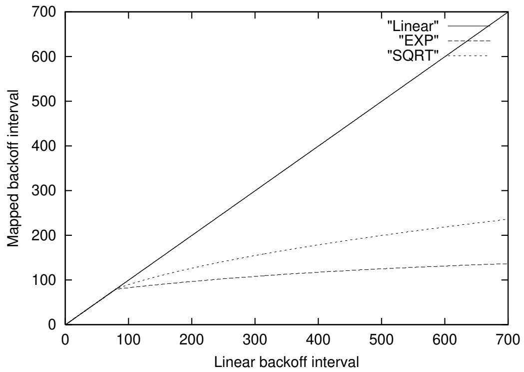



**Figure 6: Illustration of the mapping functions: For all mappings, the *Scaling_Factor* is assumed to be 0.02. For the exponential and linear mappings, the *Threshold* is 80. For exponential mapping, $K_1 = 80$ and $K_2 = 0.002$.**

## 3.3 Summary of DFS Protocol

$$\Delta = \begin{cases} \Delta - (\Delta_{current} \text{ value tagged to the transmitted packet}), & \text{if } \Delta - \Delta_{current} > 0 \\ \Delta, & \text{otherwise} \end{cases}$$

$$B_i = \gamma(\Delta)$$

**Figure 4: Recalculation procedure**

In summary, the DFS protocol behaves quite similar to IEEE 802.11, the primary difference being in the way backoff interval is calculated initially. Also, in case of the exponential and square-root mappings, the backoff interval is also updated whenever a node hears another node transmitting a packet. By appropriately calculating the backoff intervals, DFS is able to achieve much fairer allocation of bandwidth, than what is feasible using 802.11.

# 4. OBSERVATIONS

## 4.1 Virtual Clocks

Recall that with linear mapping, the backoff interval is calculated using Equations 4 and 5. Thus, the virtual clock value maintained by a node is not used in the calculation of backoff interval at all. This means that, when using the linear mapping, there is no need to tag the finish tag to the transmitted packet, or to maintain a virtual clock at the nodes. This is the approach used in our performance evaluation of DFS.

Similarly, in the exponential mapping scheme and the recalculation procedure presented in the paper, the virtual time is not used. Thus, there is no need to maintain virtual clocks in this case as well. For exponential and square-root mappings though, we need to tag $\Delta_{current}$ of the transmitted packet. Also, it should be noted that alternative recalculation procedures can be conceived which make use of the virtual time. When such procedures are used, it is necessary to maintain virtual clocks.

## 4.2 Multiple Flows Per Node

In our discussion of DFS, we assumed that only one flow exists at each node. In general, it is possible that each node may maintain multiple flows locally. In this case, we modify the DFS protocol in Section 3 as described below.

- Whenever a packet reaches the front of its flow at some node $i$, start and finish tags for the packet are calculated as described in DFS. Specifically, the start tag is set equal to the current virtual time at node $i$, and finish tag for the packet is set equal to the (start tag + *Scaling_Factor**packet length/flow weight).
- When node $i$ needs to choose the next packet that it will attempt to transmit, it chooses the packet, say $P$, with the smallest finish tag among packets at the front of all backlogged flows at node $i$. Backoff interval for packet $P$ is calculated using procedure described in Section 3. Rest of the steps for transmitting $P$ are identical to those described in DFS.

An analogous procedure has been suggested in the paper on MACAW [2], although that paper does not present a mechanism for allocating bandwidth proportional to weights of the flows.

## 4.3 Impact of Transmission Errors

In case of a wireless LAN, transmission errors can occur, resulting in packet loss. There are two issues that need to be addressed in this area:

- How to determine which packet is lost due to transmission errors.
- How to maintain fairness in presence of transmission errors, assuming that the above question can be answered satisfactorily.

We have performed evaluation of the proposed DFS scheme in presence of errors. Our simulations indicate that, in presence of errors, fairness achieved by DFS degrades (as might be expected), however, it remains fairer than IEEE 802.11. We now briefly present some preliminary ideas on addressing the above two questions:

- For the sender of a packet on the wireless channel, it is difficult to determine whether a packet was lost due to transmission errors, or due to collision with transmission by another node on the LAN.

  As discussed previously, IEEE 802.11 provides for an exchange of RTS and CTS packets that precedes the transmission of the data packet. The heuristic we propose (to be used in conjunction with DFS) is to assume that any loss of RTS or CTS packets is due to collisions, and any loss of a data or ACK packet is due to transmission errors. Clearly, RTS and CTS packets may be lost due to errors too. Assuming their loss to be due to collision results in the invocation of the collision handling procedure in DFS. Since the backoff interval chosen after the first collision of a packet is small, the cost of misinterpreting an error loss as a collision loss is not high.
- Compensation of flows: Many centralized approaches have been developed for improving fairness in presence of location-dependent errors [5, 16, 17, 18, 20]. Among these proposals, the schemes presented in [5] and [20] lend themselves well to a distributed implementation. An additional "compensating" flow at *each node*, similar to the *Long-Term Fairness Server* (LTFS) defined in [20] can be maintained in DFS. An LTFS is used to temporarily allocate additional bandwidth to compensate flows that suffer transmission errors. In the distributed case, one or more LTFS can be maintained at each node on the LAN, whereas in the centralized algorithm in [20], only the base station maintains

LTFSs. Reference [5] proposes a different mechanism, consisting of dynamic adaptation of weights by erroneous flows to increase *effort* in order to reclaim lost bandwidth. It shows that flows experiencing low error-rates can achieve long-term fairness. In [5], the amount of compensation can be limited administratively by means of a *power factor*. The idea of dynamic adaptation of weights has been implemented in DFS in [11] to achieve long-term fairness in the presence of errors.

### 4.4 Comparison of DFS and SCFQ

Note that we began with the goal of imitating SCFQ. As seen from the description of DFS, the DFS algorithm may appear to imitate SCFQ. However, there is a significant difference between the behaviors of SCFQ and DFS. Specifically, DFS can yield packet transmissions in an order that cannot possibly be obtained in the centralized implementation of SCFQ. In general, we believe that such a deviation is likely to occur when any centralized *work-conserving* scheme[5] is applied to a distributed environment.

To illustrate the difference between SCFQ and DFS, consider a system consisting of two flows (in the distributed case, the two flows reside on two different nodes). Let the weight of flow 1 be 0.1 and the weight of flow 2 be 0.5. Assume that, initially, both flows are empty. Also assume that a packet arrives on flow 1 at time 0, and a packet of the same size arrives on flow 2 at time 0.0002 second. Now, in the centralized implementation, since only flow 1 is backlogged at time 0 when using a work-conserving scheduler, the packet from flow 1 is transmitted at time 0, followed by the packet from flow 2.

In the distributed case, let us assume that the two flows reside on two different nodes. With the distributed implementation in DFS, a backoff interval of, say, 100 slots may be chosen for flow 1. Let us assume that a slot is of duration 0.00001 second. Also, assume that the *linear* mapping is being used. Now, the packet on flow 2 arrives at time 0.0002 second – by this time, flow 1's backoff interval would have counted down from 100 to 80 (because each slot is of duration 0.00001 second). Since weight of flow 2 is five times the weight of flow 1, the backoff interval chosen for the packet on flow 2 may be 20 slots. Thus, the backoff interval of flow 2 will count down from 20 to 0 before flow 1's backoff interval counts down to 0. Therefore, flow 2 will transmit a packet before flow 1 can transmit.

Clearly, the centralized and distributed implementations result in different ordering of packet transmissions. Essentially, this is because the distributed implementation is *not* work-conserving – some of the "work" is spent on performing medium access control (MAC), not transmitting packets from the flows. As seen above, the overhead incurred by MAC may allow transmission of packets which could not have been considered for transmission in the centralized case.

### 4.5 Choice of Weights

Performance of DFS depends on the weights assigned to various flows. Performance achieved by DFS changes if the absolute weights assigned to flows are changed, even while keeping the ratio of weights of different flows the same. In this paper, we do not consider the mechanism for determining appropriate weights for the flows. However, it should be noted that, larger weights result in DFS choosing smaller contention windows. Therefore, if the weights are chosen to be too large, DFS performance can degrade due to increased collisions.

### 4.6 Dynamic Adaptation of Weights

In the above discussion, we assumed that the weight of each flow is constant, and predefined. Since the protocol is fully distributed, a given node does not need to be aware of the weights of flows at other nodes. Also, the complexity of the protocol code executed at any node is independent of whether the weight assigned to a flow is a constant or changes dynamically. Thus, it is possible to use the proposed protocols in an environment where (potentially) a different weight may be chosen for each packet on a flow, *without* increasing protocol complexity. However, it should be noted that, to maintain the complexity, the weight assigned to a given packet should not be changed *after* its start and finish tags have been assigned (i.e., after it has reached the front of its flow).

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), proportional to the size of the pending packet queue for the flow, or as a function of errors experienced by the flow.

In centralized implementations of fair queueing algorithm, dynamic changes in weights can result in significant increases in time complexity of the algorithm. However, for DFS, this is not the case.

### 4.7 Adaptive DFS

DFS protocol uses three parameters: *Scaling_factor*, *CollisionWindow* and $\rho$. These parameters can be chosen adaptively to improve performance. There is a trade-off between throughput and fairness achieved by the choice of *Scaling_factor* as seen in Section 5. Since *CollisionWindow* determines the contention window values chosen after collision, a larger value of this parameter may be chosen with increased contention on the channel. Future work will consider adaptive mechanisms to choose appropriate parameters dynamically.

## 5. PERFORMANCE EVALUATION

In this section, we present performance evaluation results for the proposed DFS protocol. Performance evaluation is performed using a modified version of the *ns-2* simulator [6]. The *ns-2* simulator includes a module to simulate the DCF function in IEEE 802.11. We modified this module to simulate the proposed DFS protocol as well. The channel bandwidth is assumed to be 2 Mbps. The virtual clock is not used in the implementation as discussed in Section 4.1.

[5]When using a work-conserving server, the output channel is not kept idle if any flow is backlogged.

In the simulation environment, the number of nodes on the LAN is $n$, where we have considered $n \leq 128$. On a LAN with $n$ nodes, we set up $n/2$ flows ($n$ is always chosen to be an even number) – flow $i$ is set up from node $i$ to node $i+1$ (the nodes are numbered 0 through $n-1$). The choice of the destination nodes for the flows is somewhat arbitrary, and any destination could have been chosen for each flow without affecting the results.

Unless otherwise specified, the following assumptions are made:[6] (i) each flow is backlogged throughout the duration of the simulation. (ii) all packets on all flows contain 584 bytes.[7] (iii) *Scaling_Factor* is 0.02. (iv) *CollisionWindow* is 4 slots. (v) Sum of weights of all flows add to 1. (vi) For the *exponential* and *square-root* mapping schemes, the *Threshold* = 80. For the exponential mapping, $K_1 = 80$ and $K_2 = 0.002$. (vii) The duration of simulations is 6 seconds.

Figure 7 considers the case when the $n/2$ flows (in case of a LAN with $n$ nodes) have identical weight – the chosen weight for each flow is $2/n$. This figure plots the ratio (throughput of a flow / flow weight) for all flows – the number of nodes $n$ is different in Figures 7(a), (b) and (c). Note that the horizontal axes in Figure 7 denote the destination node for the flow whose (throughput/weight) ratio is plotted in the figure. Results are plotted for IEEE 802.11, and the DFS scheme using the *linear*, *exponential* and *square-root* mappings. The curves labelled Linear, EXP and SQRT correspond to the DFS scheme using the respective mapping schemes. Ideally, the (throughput/weight) curve should be flat, since all flows are always backlogged. Observe that the three DFS schemes do achieve a nearly flat curve. On the other hand, observe that IEEE 802.11 results in unfair performance.

For environments where all flows are always backlogged, we evaluate a *fairness index* [13] as follows, where $T_f$ denotes throughput of flow $f$, and $\phi_f$ denotes weight of flow $f$.

$$\text{fairness index} = \frac{\left(\sum_f T_f/\phi_f\right)^2}{\text{number of flows} * \sum_f (T_f/\phi_f)^2}$$

Figure 8 studies the variation in *fairness index* (as defined above) and *aggregate throughput* with the number of flows – aggregate throughput is obtained by adding the throughput of all flows. Each flow is assigned a weight of $2/n$ (with $n/2$ flows). Average throughput and average fairness index over ten runs is considered here. Observe that DFS achieves very high fairness, while fairness achieved by IEEE 802.11 is often poor. However, the aggregate throughput achieved by 802.11 may be higher. IEEE 802.11 can sometimes achieve higher throughput because the DFS scheme tends to choose greater backoff intervals than 802.11, resulting in higher overhead for DFS.

Now, when the three mappings for the DFS scheme are considered, as seen in Figure 8, the three mappings yield comparable throughput and comparable fairness. As seen later, the exponential and square-root mappings provide benefit when the backlogged flows have relatively small weights.

Figure 9 plots fairness index and aggregate throughput as a function of the *Scaling_Factor*. An average of throughput and fairness index over four runs is considered here. Here we only consider the linear mapping. In this case, six flows are simulated with weights being 1/2, 1/4, 1/8, 1/16, 1/32, and 1/32. Observe that as the *Scaling_Factor* is increased fairness increases. The throughput initially improves when the *Scaling_Factor* is increased, but then degrades after the *Scaling_Factor* is increased further. A larger *Scaling_Factor* results in large backoff intervals, leading to a greater overhead. When the *Scaling_Factor* is very small, there are too many collisions, resulting in low throughput – when the *Scaling_Factor* is increased, collisions reduce, and throughput improves. However, larger when *Scaling_Factor* is increased further, throughput degradation due to large backoff intervals starts to dominate, and the aggregate throughput decreases. Figure 9 reinforces the observation that a trade-off exists between aggregate throughput and fairness.

Now we consider the impact of differing packet sizes among flows. In Figure 10 we evaluate 3 flows each with weight 1/3, but their packet sizes are 584, 328 and 200 bytes, respectively. The figure plots (throughput/weight) for the three flows. Observe that the curve is horizontal for DFS schemes. The DFS scheme can handle packets of differing sizes without affecting fairness. We also simulated environments where packet sizes vary *within* each flow. The results are similar to those reported in Figure 10, and are omitted here for brevity.

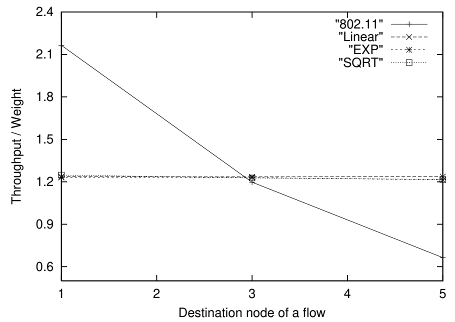
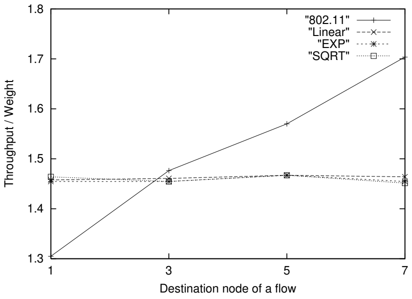



**Figure 10: Fairness with variable packet sizes**

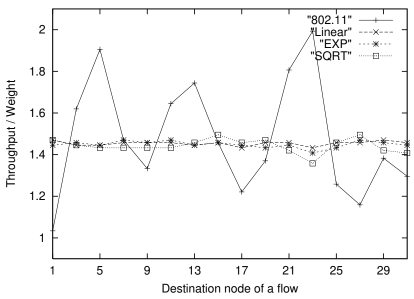
Now consider the case of four flows: flow $0 \rightarrow 1$ with weight 0.02, and flow $2 \rightarrow 3$ with a weight of 0.03, flow $4 \rightarrow 5$ with a weight 0.05 and flow $6 \rightarrow 7$ with a weight of 0.9. First assume that all four flows are always backlogged. Results for this case are shown in Figure 11 – this figure plots throughput/weight for the four flows. Observe that all three DFS mappings are fair, although linear mapping gives slightly

[6] The evaluation presented here differs somewhat from [24]. For instance, the virtual clock field is eliminated from the packet header in this paper.

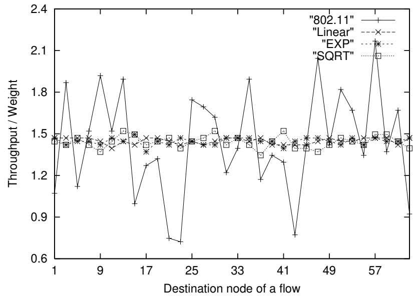
[7] 584 bytes comprise of 512 data bytes and 72 header bytes. The header bytes included in packet size are counted towards bandwidth allocated to a flow. Counting header bytes towards allocated bandwidth discourages small packets. However, note that DFS will perform just as fairly even if header bytes are not counted towards useful work performed for a flow. The exponential and square-root mappings have an extra 4 bytes in the MAC header for the $\Delta_{current}$ field. These 4 bytes are not counted in the throughput calculation for uniformity.




(a) 8 Nodes

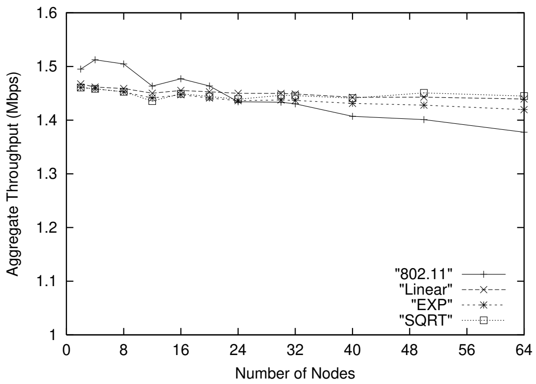



(b) 32 Nodes

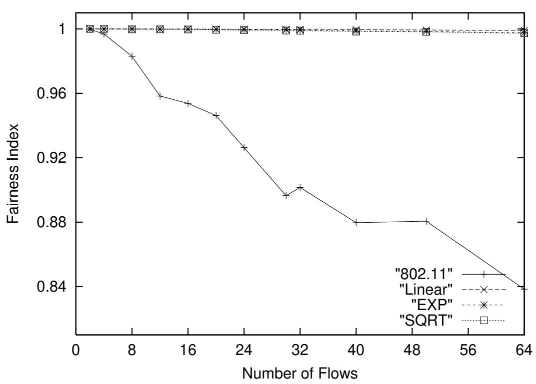



(c) 64 Nodes

**Figure 7: Comparison of IEEE 802.11 and DFS**

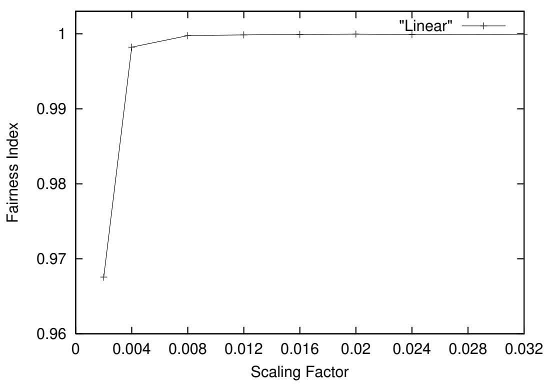



(a) Aggregate Throughput

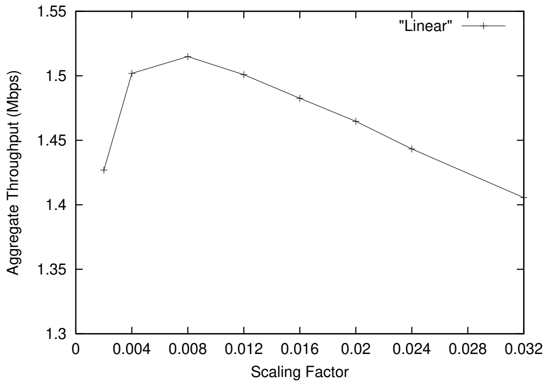



(b) Fairness Index

**Figure 8: Average aggregate throughput and fairness index versus number of flows**




(a) Fairness Index




(b) Aggregate Throughput

**Figure 9: Impact of scaling factor**

higher throughput.

Now, let us change the behavior of flow $6 \rightarrow 7$ (weight 0.9) such that it is initially on for 0.3 seconds, then off for 5.4 second, and on for remaining 0.3 seconds of simulation. Thus, this flow is on for 10% of the time. In this case, aggregate throughput achieved by the three lower weight flows using the three mapping schemes is approximately: (a) Linear: 79 Kbps, (b) Exponential: 95 Kbps, and (c) Square-root: 90 Kbps. Exponential and square-root schemes yield 20% and 14% improvement over Linear. The fairness achieved by the exponential and square-root schemes remains high, in addition to the higher throughput.

The above example illustrates that the square-root and exponential mappings can yield better throughput than the linear mapping (along with good fairness) when the aggregate weight of backlogged flows is small. On the other hand, when some backlogged flows have large weights – their backoff intervals are small and the idle time while counting down the backoff interval is bounded by the smallest backoff interval. Therefore, when at least one flow with a large weight is backlogged, the gain due to exponential and square-root mappings is not significant.

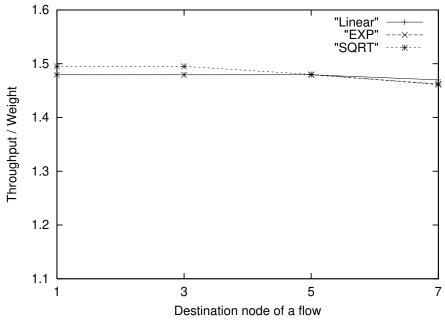



**Figure 11: Fairness with variable weight (all flows are always backlogged)**

The results reported so far evaluate fairness of the proposed algorithm over somewhat longer time intervals. We now consider fairness over shorter intervals. A variation of 802.11 referred to as 802.11_Scaled is also considered here. 802.11_Scaled chooses contention window values in the interval [0,cw], where cw is the maximum backoff interval picked by DFS after randomization. This allows us to study the impact of proportionally large windows on fairness in 802.11. Figure 12 illustrates the short-term behavior of the DFS protocol in comparison to 802.11. For 8 flows, each with weight 1/8, we count the number of packets (all packets are size 584 bytes) serviced from each flow over a window of size 0.04 second, where the window itself slides every 0.02 second. Figure 12 plots the frequency distribution of the number of packets received in a single window interval. Observe that DFS always receives either 1 or 2 packets in all intervals. 802.11 receives 0 packets in some intervals showing that some flows were put into backoff unfairly during those intervals. 802.11_Scaled performs better than 802.11 by achieving a smaller spread than 802.11. We obtained similar plot for higher number of flows as shown in [11]. In general, note that a *wider* distribution in Figure 12 is an indication of poorer short-term fairness.

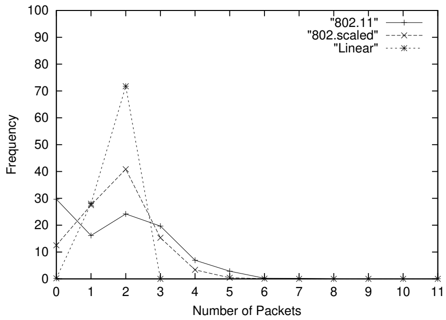



**Figure 12: Number of packets received in sliding windows of length 0.04 sec, window itself sliding every 0.02 sec**

Figure 13 shows the convergence characteristics of the fairness index. This plot is for 24 flows each with a weight of 1/24. Note that DFS converges to a high fairness index very soon, when the interval over which the fairness is evaluated is increased – [15] has considered a similar approach for evaluating short-term fairness. Observe that convergence of 802.11_Scaled is faster as compared to 802.11. This shows that the performance of 802.11 can be improved significantly by choosing proportionally large initial contention windows. Yet DFS achieves higher fairness index than 802.11 and 802.11_Scaled both. Hence, we believe that high fairness using DFS is achieved due to three factors: choice of proportionally large initial backoff intervals, choice of a small window after collision, and the fact that the initial window is chosen over an interval that is typically a small fraction of the chosen window size.

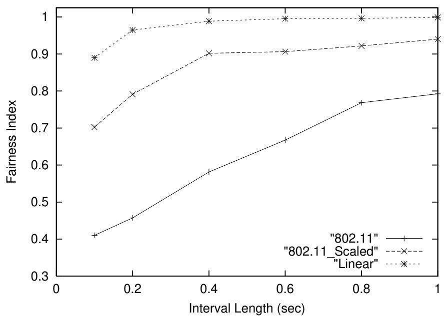



**Figure 13: Convergence of fairness index**

# 6. CONCLUSIONS

This paper considers the issue of *fair scheduling* in a wireless LAN. The objective here is to develop a *fully distributed* algorithm for scheduling packet transmissions such that different flows are allocated bandwidth in proportion of their weights. The paper proposes a Distributed Fair Scheduling (DFS) approach obtained by modifying the Distributed Coordination Function(DCF) in IEEE 802.11 standard. The similarities between DFS and DCF would make it easier to incorporate DFS in a modified version of 802.11.

Performance results show that the proposed protocol can, in fact, allocate bandwidth in proportion to the weights of the flows sharing the channel. We propose various *mappings* that can be used to choose the appropriate *backoff interval* for a packet. It is shown that all proposed mapping schemes achieve good fairness. However, the throughput achieved by the *exponential* and *square-root* mapping schemes is higher than that with *linear mapping* when the backlogged flows have low weights. In general, a trade-off exists between fairness and achievable throughput on the LAN.

Note that the ideas presented here may be applied to wired LANs as well. Also, these ideas may be extended to multi-hop wireless networks [24]. Several issues, in addition to those described in this paper, need to be investigated to achieve a better understanding of the DFS protocol.

## Acknowledgements

This work is supported in part by National Science Foundation grant number ANI-9973152. This work was partly performed during a visit by Nitin Vaidya to Microsoft Research [24]. We thank the reviewers for their useful comments.

## 7. REFERENCES

[1] J. C. R. Bennett and H. Zhang, "Wf2q: Worst-case fair weighted fair queueing," in *INFOCOM'96*, March 1996.

[2] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A media access protocol for wireless LANs," in *ACM SIGCOMM*, pp. 212–225, August 1994.

[3] G. L. Choudhury and S. S. Rappaport, "Priority access schemes using CSMA-CD," *IEEE Transactions on Communications*, vol. COM-33, pp. 620–626, July 1985.

[4] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. SIGCOMM*, September 1995.

[5] D. Eckhardt, P. Steenkiste, "Effort-limited Fair (ELF) Scheduling for Wireless Networks," *In Proc. IEEE INFOCOM 2000.*

[6] K. Fall and K. Varadhan, "*ns* Notes and documentation," tech. rep., VINT Project, UC-Berkeley and LBNL, 1997.

[7] R. Garces and J. J. Garcia-Luna-Aceves, "Near-optimum channel access protocol based on incremental collision resolution and distributed transmission queues," in *IEEE INFOCOM, San Francisco*, March-April 1998.

[8] M. Gerla, K. Tang, and R. Bagrodia, "TCP performance in wireless multi-hop networks," in *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pp. 41–50, February 1999.

[9] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *IEEE INFOCOM*, 1994.

[10] 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, pp. 690–704, October 1997.

[11] S. Gupta, "Study of distributed fair scheduling in a wireless LAN," Master of Science thesis in preparation, Texas A&M University, May 2000.

[12] IEEE, "IEEE std 802.11 - wireless LAN medium access control (MAC) and physical layer (PHY) specifications," 1997.

[13] R. Jain, G. Babic, B. Nagendra, and C. Lam, "Fairness, call establishment latency and other performance metrics," Tech. Rep. ATM_Forum/96-1173, ATM Forum Document, August 1996.

[14] S. Keshav, "On the efficient implementation of fair queueing," *Journal of Internetworking: Research and Experience*, vol. 2, pp. 57–73, September 1991.

[15] C. E. Koksal, H. I. Kassab, H. Balakrishnan, "An Analysis of Short-Term Fairness in Wireless Media Access Protocols," Extended version of short paper to appear in *Proc. ACM SIGMETRICS*, June 2000.

[16] S. Lu, T. Nandagopal, and V. Bharghavan, "A wireless fair service algorithm for packet cellular networks," in *ACM MobiCom*, 1998.

[17] T. Nandagopal, S. Lu, and V. Bharghavan, "A unified architecture for the design and evaluation of wireless fair queueing algorithms," in *ACM MobiCom*, August 1999.

[18] T. S. Ng, I. Stoica, and H. Zhang, "Packet fair queueing: Algorithms for wireless networks with location-dependent errors," in *INFOCOM*, March 1998.Fair queuing in wireless networks

[19] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, June 1993.

[20] P. Ramanathan and P. Agrawal, "Adapting packet fair queueing algorithms to wireless networks," in *ACM MobiCom*, 1998.

[21] S. M. Sharrock and D. H. Du, "Efficient CSMA/CD-based protocols for multiple priority classes," *IEEE Transactions on Computers*, vol. 38, pp. 943–954, July 1989.

[22] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," in *SIGCOMM'95, Cambridge, MA, USA*, 1995.

[23] 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.

[24] N. H. Vaidya and P. Bahl, "Fair scheduling in broadcast environments," Tech. Rep. MSR-TR-99-61, Microsoft Research, August 1999.