Distributed Fair Scheduling in a Wireless LAN

Nitin Vaidya, *Senior Member*, *IEEE*, Anurag Dugar,
Seema Gupta, and Paramvir Bahl, *Senior Member*, *IEEE*

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.

Index Terms—Medium access control, wireless networks, weighted fairness, distributed protocols.

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 and the availability of several commercial products based on this standard. Fairness is an important issue when accessing a shared wireless channel. With *fair scheduling*, different flows wishing to share the wireless channel can be allocated bandwidth in proportion to their "weights." This paper presents a *distributed* medium access control (MAC) protocol for fair scheduling in a wireless LAN (operated in an "ad hoc" mode). Although IEEE 802.11 wireless MAC 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 the proposed approach is that it can be implemented with simple modifications to IEEE 802.11.

In general, medium access control (MAC) protocols can be divided into two types: centralized and distributed. In centralized protocols, a designated host (often referred to as base station or access point) coordinates access to the wireless medium. Point Coordination Function (PCF) in IEEE 802.11 is an example of the centralized approach. 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.

This paper develops a distributed approach for fair scheduling. Much research has been performed on "fair queuing" algorithms for achieving a *fair* allocation of bandwidth on a shared link [2], [4], [14], [20]. Consider the system shown in Fig. 1, where a node maintains several *queues* (or *flows*) which store packets to be transmitted on an output link. A fair queuing 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.

Fair queuing algorithms in the literature typically attempt to approximate the Generalized Processor Sharing (GPS) discipline [20]. When using the GPS discipline, a server serves, say, n flows, each characterized by a positive *weight*; let ϕ_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 [20], if flow i is backlogged[1] throughout $[t_1, t_2]$, the following condition holds:

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

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 $[t_1, t_2]$. 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 the development of several *packet* fair queuing algorithms which approximate GPS under the constraint that each

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

- *N. Vaidya is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1308 West Main St., Urbana IL 61801. E-mail: nhv@crhc.uiuc.edu.*
- *A. Dugar is with OPNET Technologies Inc., 7255 Woodmont Avenue, Bethesda, MD 20878. E-mail: anurag_dugar@yahoo.com.*
- *S. Gupta is with Cisco Systems Inc., 510 McCarthy Blvd, Milpitas, CA 95035. E-mail: segupta@cisco.com.*
- *P. Bahl is with Microsoft Research, One Microsoft Way, Redmond, WA 98052. E-mail: bahl@microsoft.com.*

Manuscript received 22 Aug. 2003; revised 21 May 2004; accepted 12 July 2004; published online 28 Sept. 2005.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0134-0803.

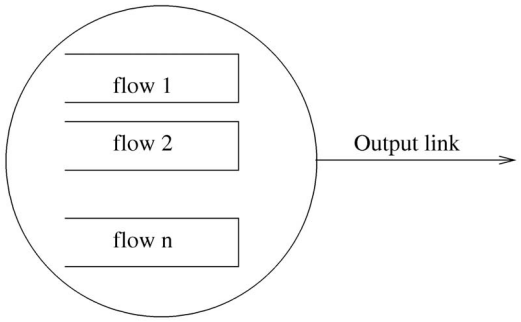


Fig. 1. A node with several flows sharing a link.

packet must be transmitted as a whole [2], [4], [14], [20]. These protocols are centralized by design, as noted above.

There has been some work on achieving fairness in wireless networks (e.g., [3], [12], [16], [17], [27], [26]). Some of the past work on incorporating fairness into distributed protocols has been limited in that these protocols attempt to provide an *equal* share of bandwidth to different nodes (essentially, node weights are implicitly assumed to be equal). Other researchers have developed protocols that take into consideration multihop network topology to achieve fairness. For instance, Luo et al. [16] have developed mechanisms that achieve weighted fairness while also trying to maximize the throughput. They proposed an interesting approach based on the notion of a flow contention graph that takes into account the topology of the network. They have also developed a topology-independent model for fair queuing [17]. The protocol proposed in this paper is relatively simple, but with relatively modest goals compared to the work of Luo et al.

In recent years, researchers have also considered the use of fair queuing in the wireless cellular environment illustrated in Fig. 2a. 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 the presence of *location-dependent* errors [19]. 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 the presence of location-dependent errors have been developed [15], [18], [19], [21]. These approaches are *centralized* and require the base station to coordinate access to the wireless channel, whereas the proposed protocol is distributed.

There has also been work on distributed protocols that takes *priorities* into account when performing medium access control [1], [25], [23], [24]. For instance, Aad and Castelluccia [1] present service differentiation mechanisms for wireless networks. Their mechanism allows a host to pick a backoff interval as a function of its priority, larger backoff intervals being used for lower priority. Our proposed fair scheduling mechanism uses a similar mechanism, but with the goal of achieving weighted fair scheduling, not priority scheduling. Our related work [22] proposed a distributed MAC protocol for wireless networks to support prioritized scheduling along with a weighted fair sharing of the bandwidth among the users belonging to the same priority level. 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.

The rest of this paper is organized as follows: Section 2 discusses some background on the SCFQ fair queuing protocol and IEEE 802.11. The proposed protocol is discussed in Section 3. An approach to improve performance of the proposed protocol is presented in Section 4 and an adaptive protocol in Section 5. Section 6 makes some interesting observations about the proposed method. Performance evaluation is presented in Section 7. Finally, conclusions are presented in Section 8.

2 Preliminaries

The objective behind this work was to develop a fair scheduling MAC protocol for a wireless LAN (illustrated in Fig. 2b), with the following properties: 1) The protocol must

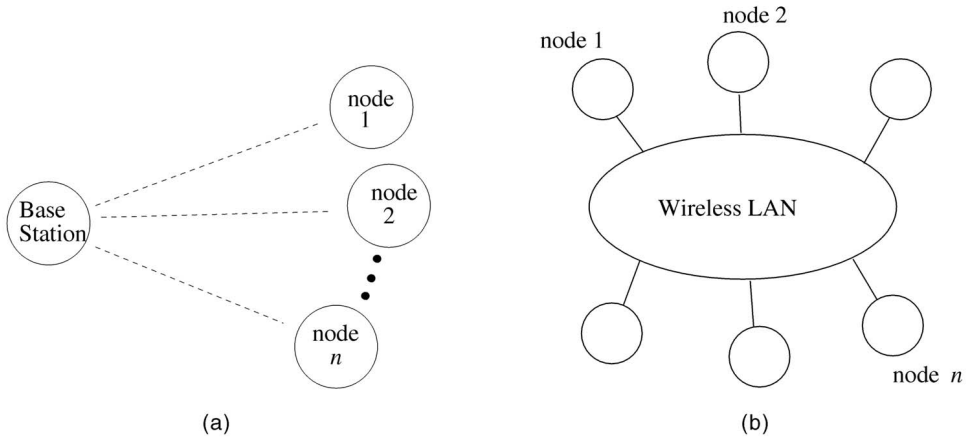


Fig. 2. Wireless environments. (a) In centralized approaches, the base station coordinates medium access. (b) In distributed approaches, all nodes have identical responsibilities.

be *fully distributed* in that no single node should have any special responsibility. 2) 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—the state of a flow includes information such as the weight of the flow, whether the flow is backlogged or not, and the time of the arrival of packets on the flow. 3) Maintain compatibility or close resemblance to an existing wireless MAC standard, to make it easier to implement the proposed protocol. The next two sections describe a *centralized* fair queuing algorithm and the IEEE 802.11 MAC protocol, which together form the basis for the proposed fair scheduling protocol.

2.1 Self-Clocked Fair Queuing (SCFQ)

The algorithm proposed here was designed in an attempt to emulate Self-Clocked Fair Queuing (SCFQ) in a distributed manner. Two important issues are worth noting here: 1) The proposed technique to implement distributed fair scheduling can also be extended to other fair queuing algorithms, such as Start-Time Fair Queuing (SFQ) [14]. 2) Although our intention was to emulate SCFQ, the distributed implementation behaves somewhat differently, as discussed later in Sections 6.1 and 6.4.

Now, we briefly describe the centralized SCFQ algorithm [13] which assumes the architecture shown in Fig. 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 kth 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 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 [14], WFQ [4], WF2Q [2], 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

$$S_i^k = v(f_i^k). \tag{1}$$

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.

2.2 IEEE 802.11 MAC: Distributed Coordination Function

The medium access control protocol specified in the 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. A 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 the size of the so-called *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. 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 nonzero, 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 destination of the packet. The destination node, on receiving the RTS, sends a Clear-to-Send (CTS) packet. Node i, on receipt of the CTS packet, transmits the data packet. The receiver node, on receipt of data, sends an acknowledgment (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 Distributed Fair Scheduling (DFS) Protocol

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

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 interframe spacing, before starting to decrement B_i. We will omit such details in this discussion. However, our simulation model does implement these details accurately.

- The DFS protocol borrows SCFQ's idea of transmitting the packet whose finish tag is the 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 the 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 6.2). Each node i maintains a local virtual clock, $v_i(t)$, where $v_i(0) = 0$. Now, P_i^k represents the kth 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 (1) 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 the appropriate choice of the *Scaling_Factor* allows us to choose a suitable scale for the virtual time:

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

- The objective of the next step is to choose a backoff interval such that a packet with a 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:

$$B_i = \left\lfloor F_i^k - v(f_i^k) \right\rfloor \text{ slots.} \quad (2)$$

  Now, observe that, since

$$F_i^k = v(f_i^k) + Scaling\_Factor * \frac{L_i^k}{\phi_i},$$

  the above expression reduces to:

$$B_i = \left\lfloor Scaling\_Factor * \frac{L_i^k}{\phi_i} \right\rfloor. \quad (3)$$

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

$$B_i = \lfloor \rho * B_i \rfloor, \quad (4)$$

  where ρ is a random variable with mean 1. In our simulations, ρ is uniformly distributed in the interval $[0.9, 1.1]$.

  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.[5] 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.

  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 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 one 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

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 larger finish tag may be transmitted before a packet with a smaller finish tag. To ensure that virtual clocks are nondecreasing, $max(v_i(t), Z)$ is used in this step. Incidentally, as discussed later in Section 6.1, DFS can be implemented *without* maintaining virtual clocks at the nodes.

5. Recall that, when the backoff interval reaches 0, a node transmits an RTS (request-to-send) packet, similar to IEEE 802.11. When two or more nodes count down backoff intervals to 0 simultaneously, their RTS packets would collide.

$$B_i = \gamma(\Delta) = \begin{cases} \Delta, & \text{if } \Delta < Threshold \\ \left\lfloor Threshold + K_1 * \left(1 - e^{-K_2 * (\Delta - Threshold)}\right) \right\rfloor, & otherwise \end{cases} \quad (5)$$

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

protocol which ensures this with high probability) must be used. Protocols for collision resolution in a wireless LAN have been proposed [7], [22]. Analogous approaches may be used in conjunction with our algorithm as well. For performance evaluation, we consider the DFS algorithm presented above without using a perfect collision resolution algorithm.

- Observe that, in DFS, the duration of the backoff interval is directly proportional to the $Scaling\_Factor$ and inversely proportional to the weight of a flow. When the weights of *backlogged* flows are small or the $Scaling\_Factor$ is chosen to 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 two solutions. In one of these solutions, we use mapping schemes to compress the backoff intervals. In this solution the $Scaling\_Factor$ is constant and predefined. In the other solution, we use an adaptive scheme to dynamically choose the $Scaling\_Factor$ as a function of contention for the channel.

4 Mapping Schemes

We will refer to the scheme presented above for calculating the backoff interval as the *linear* scheme (or linear mapping). From (2) and (3), 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 a function of the finish tag using some mapping schemes as follows (other alternatives are also possible).

4.1 Exponential Mapping Scheme

Let Δ denote the backoff interval obtained in (4) 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 Fig. 3. In the definition of $\gamma(\Delta)$ in Fig. 3, note that *Threshold*, K_1, and K_2 are constant parameters.

Use of the γ function has the impact of compressing large Δ 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 1,000 bytes and $Scaling\_Factor = 1/100$, the linear approach may yield backoff intervals of 1,000 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 $\gamma(1,000) = 147$ 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, $\gamma(990) = \gamma(1,000) = 147$; therefore, two nodes which simultaneously begin counting down from initial backoff intervals of 990 and 1,000 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 of such additional collisions, when defining γ 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* may result in better throughput but poorer fairness, depending upon the network conditions. On the other hand, 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.

4.1.1 Recalculation of Backoff Intervals

Unlike the case of linear mapping, additional care needs to be taken to ensure fair allocation in the case of the 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.

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

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

Fig. 4. Recalculation procedure.

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 1,000 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., ρ) 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.[6] 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 $Threshold = 80$ slots, $K_1 = 80$, and $K_2 = 0.002$, then flow 1 will continue to use a 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. When using the recalculation procedure, the Δ 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 Δ 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 Δ and B_i for its pending packet (if any), as shown in Fig. 4.

The final step in Fig. 4 recalculates the 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 Δ 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 Δ 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, the most recently calculated values of Δ 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 a backoff interval of 97 slots and $\Delta = 200$ slots and eventually transmit a packet. This packet would then have been tagged with its $\Delta_{current} = 200$.

4.2 Other Mappings

In general, any increasing function can be used to map Δ values to backoff intervals, similarly to the γ 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 Δ 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 Fig. 5. We will refer to the mapping in Fig. 5 as the *square-root mapping*. The 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 $\gamma(\Delta)$.

Fig. 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.

5 Adaptive DFS

The performance of DFS depends on the $Scaling\_Factor$ chosen and the weights assigned to various flows. In the earlier discussion in this paper, we had chosen a static $Scaling\_Factor$ and had assigned static weights to the flows.

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

Fig. 5. Function Ψ.

6. In reality, due to randomization, the backoff intervals will not be constant for a given flow.

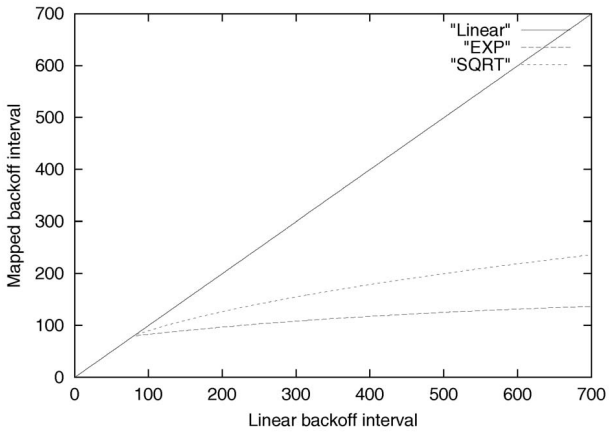


Fig. 6. Illustration of the mapping functions: For all mappings, the $Scaling\_Factor$ is assumed to be 0.02. For the exponential and linear mappings, the $Threshold$ is 80. For exponential mapping, $K_1 = 80$ and $K_2 = 0.002$.

The choice of weights for the flows and the $Scaling\_Factor$ can significantly affect the performance of DFS. 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. On the other hand, smaller weights result in DFS choosing larger contention windows. Therefore, if the weights are chosen to be too small, DFS performance can degrade due to increased overhead. Similarly, when the $Scaling\_Factor$ is chosen to be large, it results in DFS choosing large contention windows, leading to a greater overhead. On the other hand, when the $Scaling\_Factor$ is chosen to be small, it results in DFS choosing small contention windows, thereby resulting in an increase in the probability of collisions. Since the load on the network, as well as the number of nodes in the Wireless LAN, can vary dynamically, it is difficult to choose an appropriate value of the $Scaling\_Factor$ and appropriate weights for the flows that would result in a good performance under all load conditions. In this section, we present a scheme to dynamically adapt the $Scaling\_Factor$ as a function of the contention for the channel. In principle, although it is possible to utilize nonlinear mappings (e.g., exponential) mapping in conjunction with the adaptive DFS, here we focus only on the linear mapping.

5.1 Dynamic Adaptation of Scaling_Factor

For time-varying network conditions, dynamic adaptation of the $Scaling\_Factor$ is useful. In the proposed scheme, when the contention for the channel increases (indicated by collisions on the channel), the $Scaling\_Factor$ is increased to reduce the possibilty of collisions. On the other hand, when there is low contention for the channel, the $Scaling\_Factor$ is reduced to avoid large backoff intervals and long durations of idle time, thereby improving the aggregate throughput. The dynamic adaptation of the $Scaling\_Factor$ also obviates the need for any mapping scheme to compress the backoff interval to achieve a better performance. In the dynamic adaptation scheme, when a node gets to transmit, it piggybacks its $Scaling\_Factor_i$ on the data packet. All other nodes in the Wireless LAN snoop on the piggybacked $Scaling\_Factor$ value and adjust their $Scaling\_Factor$ accordingly so that all nodes have a unified view of the contention for the channel bandwidth.

We now describe this scheme for dynamic adaptation of the $Scaling\_Factor$ in more detail. Every node in the network maintains a *CollisionCounter* (as already explained in the earlier discussion of the DFS scheme) and a *SuccessCounter*. A node i uses these two variables in the following manner:

- Before node i is about to transmit a data packet (after receiving the CTS):
  - It resets $CollisionCounter_i$ to 0.
  - It increments $SuccessCounter_i$ by 1.
  - If $SuccessCounter_i$ exceeds a certain predefined threshold, say $Success\_Threshold$, it resets $SuccessCounter_i$ to 0 and decreases the $Scaling\_Factor$ by a predefined multiplicative decrement factor. $Success\_Threshold$, together with the decrement factor, controls how gradually the $Scaling\_Factor$ would be decreased.
  - It then piggybacks the $Scaling\_Factor$ on the data packet to be transmitted.
- After node i suffers a collision:
  - It resets $SuccessCounter_i$ to 0.
  - It increments $CollisionCounter_i$ by 1.
  - It chooses a new B_i uniformly distributed in $\left[1, 2^{CollisionCounter-1} * CollisionWindow\right]$, where *CollisionWindow* is a constant parameter.
  - If the $CollisionCounter_i > 1$, it increases the $Scaling\_Factor$ by a certain predefined multiplicative increment factor.[7]

Whenever a node i transmits a data packet, it piggybacks its $Scaling\_Factor_i$ on the data packet. All other nodes in the Wireless LAN snoop on the piggybacked $Scaling\_Factor$ value. Specifically, if a node, say node j, finds that the piggybacked $Scaling\_Factor$ is different from its $Scaling\_Factor$, it takes the following actions:

- If $CollisionCounter_j = 0$,
  - If node j was backlogged, it updates its remaining slots to be counted down (say rs_j) as follows:

$$rs_j = \left\lfloor rs_j * Scaling\_Factor_i / Scaling\_Factor_j \right\rfloor, \tag{6}$$

  - Node j sets $Scaling\_Factor_j = Scaling\_Factor_i$.
  - Node j resets $SuccessCounter_j$ to 0.
- If $CollisionCounter_j = 1$,
  - Node j sets $Scaling\_Factor_j = Scaling\_Factor_i$.

7. A node increments the $Scaling\_Factor$ only when the $CollisionCounter > 1$ because the first collision for a given packet is not necessarily an indication of heavy contention for the channel. However, when a node suffers a series of collisions for the same packet (in which case, its *CollisionCounter* would be > 1), it is likely that there is heavy contention for the channel.

- If $CollisionCounter_j > 1$,
  - If $Scaling\_Factor_j < Scaling\_Factor_i$, node j sets $Scaling\_Factor_j = Scaling\_Factor_i$. Else $Scaling\_Factor_j$ remains unchanged.

We study the performance of Adaptive DFS in the performance evaluation section.

6 Observations

6.1 Virtual Clocks

Recall that, with linear mapping, the backoff interval is calculated using (3) and (4). Thus, the virtual clock value maintained by a node is not used in the calculation of the 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 the evaluation of DFS. For exponential and square-root mappings though, we need to tag $\Delta_{current}$ of the transmitted packet. 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. However, 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.

6.2 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 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 the 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. The backoff interval for packet P is calculated using procedure described in Section 3. The 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 [3], although that paper does not present a mechanism for allocating bandwidth proportional to weights of the flows.

6.3 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: 1) How to determine which packet is lost due to transmission errors. 2) 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 the presence of errors. Our simulations indicate that, in the 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 data or ACK packet is due to transmission errors. Clearly, RTS and CTS packets may be lost due to errors, too. Assume 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 the presence of location-dependent errors [15], [18], [19], [21]. Among these proposals, the schemes presented in [5] and [21] lend themselves well to a distributed implementation. An additional "compensating" flow at *each node*, similar to the *Long-Term Fairness Server* (LTFS) defined in [21] 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 [21], 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 [10] to achieve long-term fairness in the presence of errors.

6.4 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[8] 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.

7 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 6.1.

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 $2i$ to node $2i+1$ (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.

In our simulations, each flow is assigned a fixed weight. We assume that, in a practical implementation, the weights will be assigned by an upper layer of the protocol stack, with the job of the MAC protocol being to schedule packet transmissions such that weighted fairness is achieved. When the MAC protocol fails in its task (for instance, perhaps due to transmission errors), the upper layers may potentially adapt the weights to achieve desired bandwidth distribution [10]. Such an adaptation of weights is beyond the scope of this paper.

7.1 DFS with Static Parameters

Here, we present the performance evaluation results for the proposed DFS protocol with static $Scaling\_Factor$. Unless otherwise specified, the following assumptions are made:[9]

1. Each flow is backlogged throughout the duration of the simulation.
2. All packets on all flows contain 584 bytes.[10]
3. $Scaling\_Factor$ is 0.02.
4. *CollisionWindow* is four slots.
5. For the *exponential* and *square-root* mapping schemes, *Threshold* = 80. For the exponential mapping, $K_1 = 80$ and $K_2 = 0.002$.
6. The duration of simulations is 6 seconds.

Fig. 7 considers the case when the $n/2$ flows (in the case of a LAN with n nodes) have identical weight. The chosen weight for each flow is $2/n$ (this choice is arbitrary, and the results hold for other choices too, except when the chosen weights are very large). This figure plots the ratio (throughput of a flow/flow weight) for all flows. The number of nodes n is different in Figs. 7a, 7b, and 7c. Note that the horizontal axes in Fig. 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 curve labeled Linear, EXP, and SQRT corresponds 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* [11] as follows, where T_f denotes throughput of flow f, and ϕ_f denotes weight of flow f:

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

Fig. 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 averaged fairness index over 10 runs are considered here. Observe that DFS

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

9. The evaluation presented here differs somewhat from [8]. There are some differences in their implementations, such as the virtual clock field is eliminated from the DFS header in this work.

10. 584 bytes is comprised of 512 data bytes and 72 bytes of UDP, IP, and MAC headers. 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.

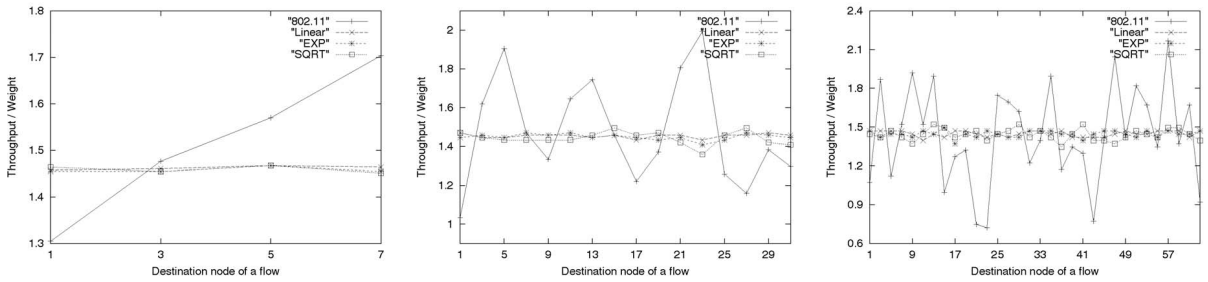


Fig. 7. Comparison of IEEE 802.11 and DFS. (a) Eight nodes. (b) 32 nodes. (c) 64 nodes.

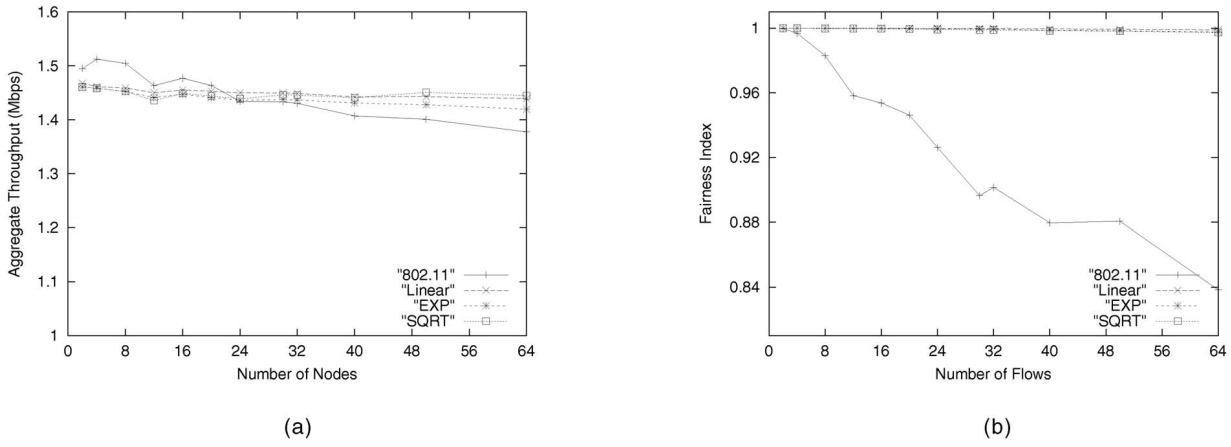


Fig. 8 Average aggregate throughput and fairness index versus number of flows. (a) Aggregate throughput. (b) Fairness index.

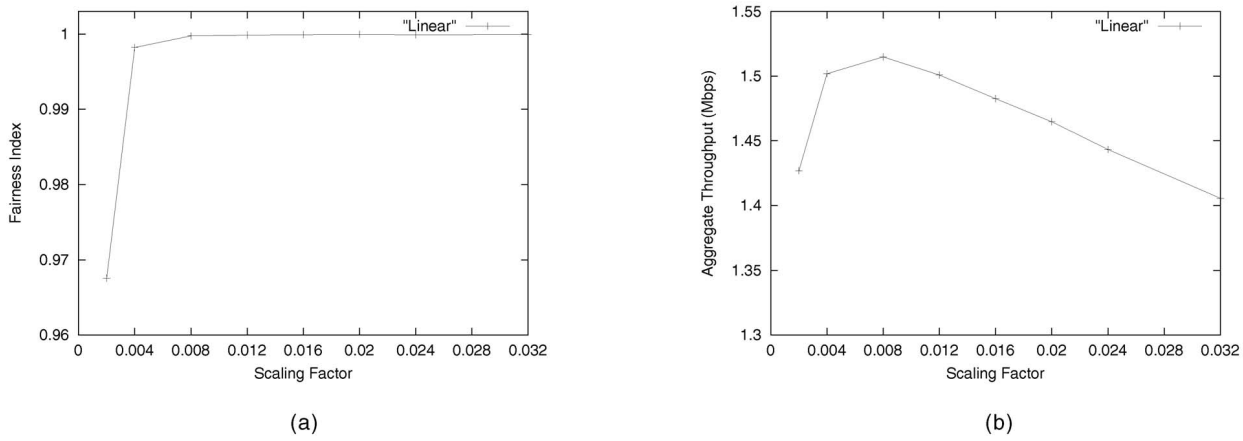


Fig. 9 Impact of scaling factor. (a) Fairness index. (b) Aggregate throughput.

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 results in a greater 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 Fig. 8, the three mappings yield comparable throughput and comparable fairness. As seen later, the exponential and square-root mappings provide a benefit when the backlogged flows have relatively small weights—particularly when the weights of backlogged flows add up to much less than 1. Fig. 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—results for other mapping are analogous. 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

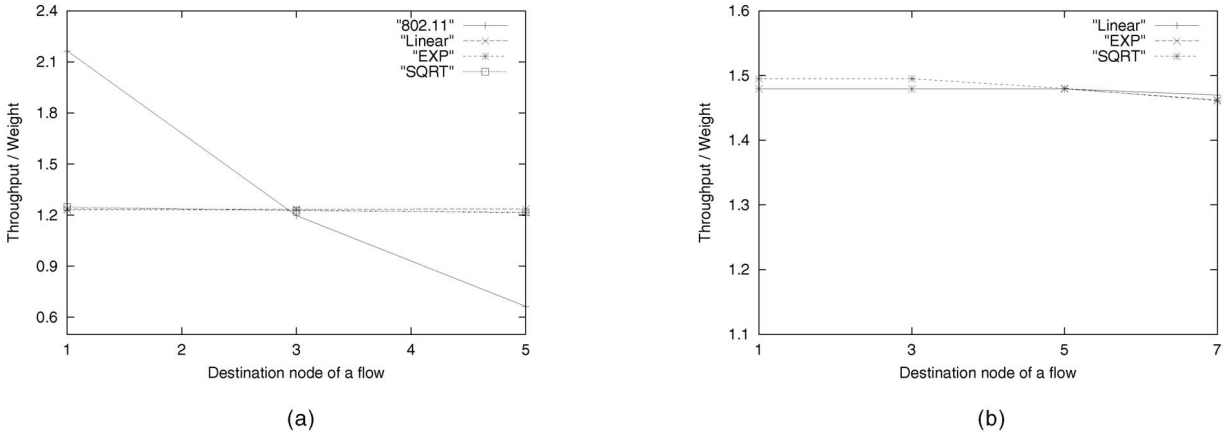


Fig. 10 Fairness results. (a) Fairness with variable packet sizes. (b) Fairness with variable weight.

increased, collisions reduce and throughput improves. However, when $Scaling\_Factor$ is increased further, throughput degradation due to large backoff intervals starts to dominate, and the aggregate throughput decreases. Fig. 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 Fig. 10a, we evaluate three 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 in Fig. 10a and are omitted for brevity.

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 Fig. 10b. This figure plots throughput/weight for the four flows. Observe that all three DFS mappings are fair, although linear mapping gives slightly higher throughput.

Next, assume that flow $6 \rightarrow 7$ with weight 0.9 is an on-off source, with on-off periods of 0.3 and 5.4 seconds, respectively. This flow coexists with the other three lower weight flows listed above such that, for a simulation of 6 seconds, flow $6 \rightarrow 7$ is on for 10 percent of the time. The aggregate throughput achieved by the three lower weight flows by the three mapping schemes is approximately: 1) Linear: 79 Kbps, 2) Exponential: 95 Kbps, and 3) Square-root: 90 Kbps. The exponential and square-root schemes yield 20 and 14 percent improvement over Linear. The fairness achieved by the exponential and square-root schemes remains high, in addition to the higher throughput. We calculated the fairness index for each mapping, over the different subintervals during which the set of backlogged flows remained constant; the fairness indices for all three mappings were over 0.999 in all cases. 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.

The results reported so far essentially evaluate the long-term fairness of the proposed algorithm. 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. Fig. 11 illustrates the short-term behavior of the DFS protocol in comparison to 802.11. We count the number of packets (all packets are the same size in this case) serviced from each flow over a window of size 0.04 second, where the window itself slides every 0.02 second. Fig. 11 plots the frequency distribution of the number of packets received by 8 flows, each with a weight of 1/8. Observe that DFS always receives either one or two packets in all intervals. 802.11 receives zero 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 a similar plot for higher number of flows as shown in [10].

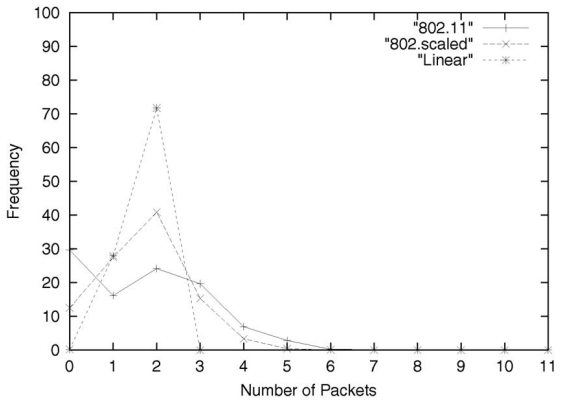


Fig. 11. Number of packets received per sliding window of 0.04 second.

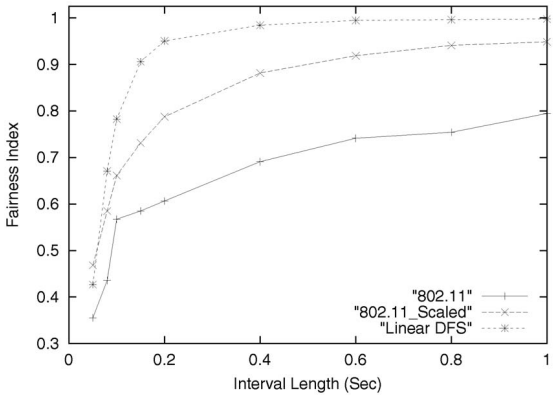


Fig. 12. Convergence of the fairness index over the short-term.

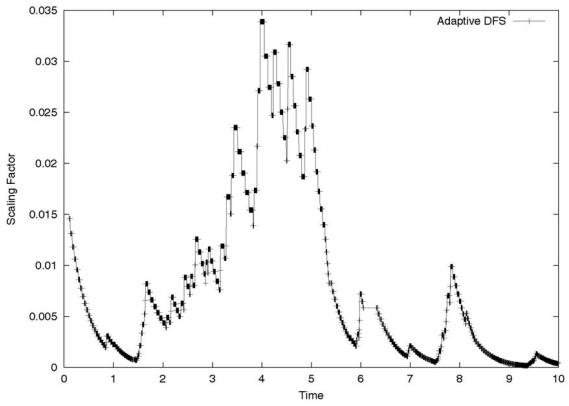


Fig. 13. Dynamic adaptation of the $Scaling\_Factor$.

Fig. 12 shows the convergence of fairness index over the short-term. This plot is for 24 flows, each with a weight of 1/24. Note that DFS converges to the unit fairness index value very soon. The 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 the choice of proportionally large initial contention windows. Yet DFS achieves a higher fairness index than 802.11 and 802.11_Scaled. Hence, the key to fairness in DFS is the choice of proportionally large initial backoff intervals and the choice of a small window after collision.

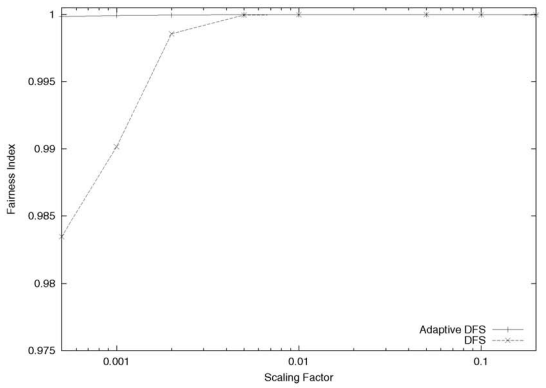
7.2 Adaptive DFS

Here, we present the performance evaluation results for the case when the DFS protocol dynamically adjusts the $Scaling\_Factor$ according to the contention for the channel. We also compare these results with the results for the DFS protocol (linear mapping), which uses a static $Scaling\_Factor$. Unless otherwise specified, the following assumptions are made:

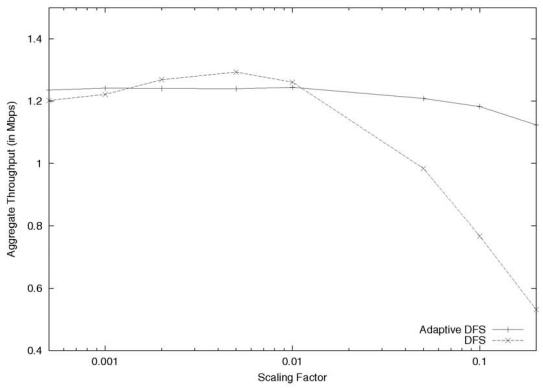
1. All packets on all flows contain 500 data bytes.
2. *Success_Threshold* is three packets.
3. Whenever the $Scaling\_Factor$ has to be increased, it is increased by 25 percent of the previous value. Whenever the $Scaling\_Factor$ has to be decreased, it is decreased by 10 percent of the previous value.
4. *CollisionWindow* (used in calculating the backoff interval after suffering a collision) is four slots.
5. The duration of simulations is 10 seconds.

Fig. 13 plots the piggybacked $Scaling\_Factor$ as a function of time. Here, we consider the case when there are 10 flows, each with a weight 0.2. Each of these flows has an on-off traffic source. The initial $Scaling\_Factor$ is 0.015. As can be observed from Fig. 13, the $Scaling\_Factor$ dynamically adapts itself according to the contention for the channel. When there is little contention for the channel, the $Scaling\_Factor$ drops down in order to reduce the overhead and long durations of idle time. However, if the contention increases on the channel, the $Scaling\_Factor$ increases to reduce the probability of collisions.

As we had discussed earlier, it is difficult to choose an appropriate value of the $Scaling\_Factor$ and appropriate weights for the flows that would result in a good performance under all load conditions. The traffic load on a Wireless LAN can vary dynamically. A given $Scaling\_Factor$ which is appropriate for a certain load condition may not work as well for other load conditions. Fig. 14 studies how the choice of the $Scaling\_Factor$ affects the performance of DFS (with static $Scaling\_Factor$) and Adaptive DFS for a static assignment of weights to



(a)



(b)

Fig. 14. Comparison of DFS and Adaptive DFS. (a) Fairness index. (b) Aggregate throughput.

the flows. Fig. 14a plots the variation in *fairness index* and Fig. 14b plots the variation in *aggregate throughput* as a function of the chosen $Scaling\_Factor$.

Here, we consider the case when there are 10 flows, each with a weight 0.05. Each of these flows has a CBR traffic source which is generating traffic at the rate of 0.2Mbps. Both DFS and Adaptive DFS start with the same initial $Scaling\_Factor$. In DFS, the $Scaling\_Factor$ remains constant throughout the entire duration of simulation, whereas in Adaptive DFS, the $Scaling\_Factor$ dynamically adapts as per the contention for the channel. Here, we plot the aggregate throughput and the fairness index achieved by DFS and Adaptive DFS as a function of the $Scaling\_Factor$. As can be observed from Fig. 14, when the $Scaling\_Factor$ is chosen to be very small, the fairness index degrades in case of DFS. In DFS, when the $Scaling\_Factor$ is very small, there is an increased likelihood of collisions which can potentially result in access priority reversals and unfairness toward colliding nodes. When the $Scaling\_Factor$ is chosen to be large, the fairness index achieved by DFS improves, but, at the same time, the aggregate throughput degrades. In DFS, large $Scaling\_Factor$ results in large backoff intervals, leading to a greater overhead. Adaptive DFS, on the other hand, dynamically adapts the $Scaling\_Factor$ and operates around the optimum $Scaling\_Factor$ under the present load conditions. Hence, irrespective of the choice of weights for the flows or the choice of the initial $Scaling\_Factor$, Adaptive DFS quickly converges to yield a good performance (both in terms of fairness and aggregate throughput).

8 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 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. We also propose a scheme for dynamic adaptation of the $Scaling\_Factor$ which allows us to achieve good performance irrespective of the choice of the initial $Scaling\_Factor$ or the assignment of weights to the flows.

Acknowledgments

This work is supported in part by the US National Science Foundation and Microsoft Research. This paper was presented in part at Mobicom 2000.

References

[1] I. Aad and C. Castelluccia, "Differentiation Mechanisms for IEEE 802.11," *Proc. INFOCOM 2001 Conf.,* 2001.
[2] J.C.R. Bennett and H. Zhang, "Wf2q: Worst-Case Fair Weighted Fair Queueing," *Proc. INFOCOM '96 Conf.,* Mar. 1996.
[3] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A Media Access Protocol for Wireless LANs," *Proc. ACM SIGCOMM Conf.,* pp. 212-225, Aug. 1994.
[4] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. SIGCOMM Conf.,* Sept. 1995.
[5] D. Eckhardt and P. Steenkiste, "Effort-Limited Fair (ELF) Scheduling for Wireless Networks," *Proc. IEEE INFOCOM 2000 Conf.,* 2000.
[6] K. Fall and K. Varadhan, "*ns* Notes and Documentation," technical report, VINT Project, Univ. of Califonia at Berkeley and Lawrence Berkeley Nat'l Laboratory, 1997.
[7] R. Garces and J.J. Garcia-Luna-Aceves, "Near-Optimum Channel Access Protocol Based on Incremental Collision Resolution and Distributed Transmission Queues," *Proc. IEEE INFOCOM Conf.,* Mar.-Apr. 1998.
[8] N.H. Vaidya and P. Bahl, "Fair Scheduling In Broadcast Environments," Technical Report MSR-TR-99-61, Microsoft Research, Aug. 1999.
[9] N.H. Vaidya, P. Bahl, and S. Gupta, "Fair Scheduling in a Wireless LAN," technical report, Texas A&M Univ., Apr. 2000.
[10] S. Gupta, "Study of Distributed Fair Scheduling in a Wireless LAN," Master of Science thesis, Texas A&M Univ., May 2000.
[11] R. Jain, G. Babic, B. Nagendra, and C. Lam, "Fairness, Call Establishment Latency and Other Performance Metrics," Technical Report ATM_Forum/96-1173, ATM Forum Document, Aug. 1996.
[12] M. Gerla, K. Tang, and R. Bagrodia, "TCP Performance in Wireless Multihop Networks," *Proc. IEEE Workshop Mobile Computing Systems and Applications (WMCSA),* pp. 41-50, Feb. 1999.
[13] S.J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," *Proc. IEEE INFOCOM Conf.,* 1994.
[14] P. Goyal, H.M. Vin, and H. Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *IEEE/ACM Trans. Networking,* vol. 5, pp. 690-704, Oct. 1997.
[15] S. Lu, T. Nandagopal, and V. Bharghavan, "A Wireless Fair Service Algorithm for Packet Cellular Networks," *Proc. ACM MobiCom Conf.,* 1998.
[16] H. Luo, S. Lu, V. Bharghavan, "A New Model for Packet Scheduling in Multihop Wireless Networks," *Proc. ACM MOBICOM Conf.,* Aug. 2000.
[17] H. Luo and S. Lu, "A Topology-Independent Fair Queueing Model in Ad Hoc Wireless Networks," *Proc. IEEE Int'l Conf. Network Protocol,* 2000.
[18] T. Nandagopal, S. Lu, and V. Bharghavan, "A Unified Architecture for the Design and Evaluation of Wireless Fair Queueing Algorithms," *Proc. ACM MobiCom Conf.,* Aug. 1999.
[19] T.S. Ng, I. Stoica, and H. Zhang, "Packet Fair Queueing: Algorithms for Wireless Networks with Location-Dependent Errors," *Proc. INFOCOM Conf.,* Mar. 1998.
[20] 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 Trans. Networking,* vol. 1, June 1993.
[21] P. Ramanathan and P. Agrawal, "Adapting Packet Fair Queueing Algorithms to Wireless Networks," *Proc. ACM MobiCom Conf.,* 1998.
[22] A. Dugar, N.H. Vaidya, and P. Bahl, "Priority and Fair Scheduling in a Wireless LAN," *Proc. IEEE Military Comm. Conf. (MILCOM),* Oct. 2001.
[23] J.L. Sobrinho and A.S. Krishnakumar, "Real-Time Traffic over the IEEE 802.11 Medium Access Control Layer," *Bell Labs Technical J.,* pp. 172-187, Autumn 1996.
[24] K. Ramamritham and W. Zhao, "Virtual Time CSMA Protocols for Hard Real-Time Communication," *IEEE Trans. Software Eng.,* vol. 13, no. 8, pp. 938-952, Aug. 1987.
[25] S.M. Sharrock and D.H. Du, "Efficient CSMA/CD-Based Protocols for Multiple Priority Classes," *IEEE Trans. Computers,* vol. 38, no. 7, pp. 943-954, July 1989.
[26] L. Tassiulas and S. Sarkar, "Maxmin Fair Scheduling in Wireless Networks," *Proc. IEEE INFOCOM Conf.,* pp. 763-772, 2002.
[27] X. Wu, C. Yuen, Y. Gao, H. Wu, B. Li, "Fair Scheduling with Bottleneck Consideration in Wireless Ad-Hoc Networks," *Proc. IEEE Int'l Conf. Computer Comm. and Networks (ICCCN),* pp. 568-572, Oct. 2001.

Nitin Vaidya received the PhD degree from the University of Massachusetts at Amherst. He is presently an associate professor of electrical and computer engineering at the University of Illinois at Urbana-Champaign (UIUC). He has held visiting positions at Microsoft Research, Sun Microsystems, and the Indian Institute of Technology-Bombay. He is a recipient of a CAREER award from the US National Science Foundation. He served as program cochair for 2003 ACM MobiCom and general chair for 2001 ACM MobiHoc. He presently serves as editor-in-chief the *IEEE Transactions on Mobile Computing*. He is a senior member of the IEEE and a member of the IEEE Computer Society.

Anurag Dugar graduated from the National Institute of Technology Karnataka, India, in electronics and communication engineering. He received the MS degree in computer engineering from Texas A&M University in 2001. He was a research assistant in the Mobile Computing and Networking Group in the Department of Computer Science at Texas A&M University. He is currently working as a senior software engineer in the Modeling Research and Development Division at OPNET Technologies, Bethesda, Maryland. At OPNET Technologies, he has been engaged in developing features in OPNET's suite of Intelligent Network Management Solutions.

Seema Gupta graduated from the Indian Institute of Technology, New Delhi, in mathematics and computer applications in 1998. She received the MS degree in computer science from Texas A&M University in 2000. Since the summer of 2000, she had worked as a software engineer with Cisco Systems, San Jose, California. She worked on several projects in the areas of network security, AAA (Authentication Authorization Accounting), and intelligent edge subscriber services.

Paramvir Bahl received the PhD degree in computer systems engineering from the University of Massachusetts Amherst. He is a senior researcher and manager of the Networking Research Group at Microsoft Research. Some of his seminal research includes: WiLIB, a general purpose programming interface for wireless network cards; RADAR, a signal-strength based indoor user-location determination system; CHOICE, an edge-server based public area wireless network; and UCOM, a multiradio wireless system. Currently, he leads MESH, a community networking and residential broadband access network project, and NetHealth, an enterprise and home network self-managing diagnostic system. Several of his ideas are incorporated into Microsoft's core Windows Operating System product. In addition to building systems, he has authored more than 65 scientific papers, 45 issued and pending patent applications, and a book chapter. He is the founder and chairman of ACM SIGMOBILE, the founder and past editor-in-chief of *ACM Mobile Computing and Communications Review* (1996-2001), and the founder of *ACM/USENIX MobiSys*. He is a fellow of the ACM, an IEEE senior member, and past president of the electrical engineering honor society Eta Kappa Nu-Zeta Pi.