Grant to Send: Fairness and Isolation in Low-Power Wireless

Technical Report SING-06-01

Jung Il Choi and Philip Levis
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Abstract

We present the Fair Waiting Protocol (FWP), which isolates competing multihop wireless protocols. Intended for wireless sensornets, FWP sits on top of a basic CSMA data link layer. When a multihop protocol transmits a FWP packet, it embeds an interval of post-transmission exclusive access. During this period, only the packet's recipient may transmit. Where an RTS/CTS protocol requests permission for its transmitter to send, FWP grants permission to someone else by transmitting to them. FWP uses grant durations to fairly allocate the channel in a distributed fashion. FWP is 270 lines of nesC code, has a single byte header, and requires 3 bytes of state per protocol. Through simulation and real traffic loads on a testbed, we show that with these meager resources, FWP enables multihop protocols to easily and cooperatively share the channel. In the case of a collection protocol, this can improve its goodput by 30% while simultaneously improving its energy efficiency.

1. Introduction

Deciding when to transmit a packet is one of the fundamental challenges of wireless networks. This challenge is in part due to the inability for a transmitter to be able to observe what happens at a receiver. As communication is not transitive yet is through a shared medium, wireless MAC protocols seek to establish simple mechanisms for distributed collaboration. Typical CSMA layers use a single bit – am I transmitting? – to communicate, while RTS/CTS uses complete packet exchanges.

One domain where the problem is particularly acute is wireless sensornets. Their low-speed radios and tight energy budgets make every packet precious and idle listening time a luxury. In sensornets, colliding packets represent wasted energy, rather than wasted bandwidth, and higher goodput means less time is spent awake. The importance of optimizing packet delivery in this domain has led to a plenitude of MAC protocols [21, 25, 32, 39]. As most MACs provide only a simple datapath, typically network protocols use higher-level mechanisms in order to increase their efficiency, such as rate-control to prevent self-interference [12, 36].

But network-layer schemes assume that the network protocol operates in a vacuum. This approach is suitable in Internet systems, where there is a single network protocol and a small number of transport protocols (i.e., TCP, DCCP, UDP). In contrast, a sensornet can have a complex assortment of network protocols, such as collection [12, 24, 38], dissemination [11, 33, 34], logical coordinate routing [20], local aggregation [26], network aggregation [13, 18], and flooding [19]. Self-interference is only part of the problem: other protocols can just as easily interfere.

Current sensornet operating systems take a first-come, first-served approach to channel allocation. One protocol can consume a lion's share of the bandwidth. Small in-memory queues exacerbate this problem. For example, a common TinyOS routing protocol, MintRoute [37], can collapse under heavy load when route update packets are dropped from full transmit queues [8]. In addition to interference, protocols can also fail due to unfair allocation of OS resources and the wireless channel.

The motivation for this paper is the observation that for a sensornet to be able to efficiently support multiple network protocols, it must provide two key properties:

Isolation: The network should minimize inter-protocol collisions. From a protocol's perspective, when it transmits it is the sole user of the network. Other protocols may reduce how often it may transmit, but its opportunities are isolated from interference. Isolation simplifies protocol implementations and prevents one misbehaving protocol from sabotaging the entire network.

Fairness: Isolation places requirements on the wireless channel during transmissions; fairness places requirements on how often a protocol can transmit. Providing

fair channel allocation between protocols further protects them from one another, promising that each can operate in the face of wild variations in competing load.

Section 2 proposes a novel protocol abstraction, *grant-to-send* (GTS), which meets these requirements. It describes Fair Waiting Protocol (FWP), a GTS protocol that sits between a CSMA MAC and network protocols, introducing a single byte header to every packet. This header specifies a time for which, after the packet is transmitted, no node except the recipient may transmit. Unlike RTS/CTS, where a packet exchange requests the channel for a transmitter, in FWP a node grants the channel to a receiver. Network protocols can use this mechanism in several ways in order to improve their performance, such as clearing a route and scheduling data bursts. Furthermore, these quiet times can be used to tabulate how long protocols have used the channel, allowing FWP to allocate it fairly between them.

Sections 3-4 evaluate FWP in small single-hop networks of low-power sensor nodes ("motes") and on the Motelab testbed at Harvard University [35]. FWP is approximately 270 lines of nesC code, introduces a single byte header on packets, and requires $2 + 3p$ bytes of state, where p is the number of protocols the system supports. With these meager requirements it can provide inter-node fairness greater than 0.98 within single-hop cells and a median fairness of 0.97 in large multihop deployments. Its inter-protocol fairness is greater than 0.98 in all cases. Furthermore, by isolating protocols FWP increases the goodput of a tree collection protocol by up to 30% while simultaneously improving its energy efficiency by up to 10%. As traffic loads increasingly vary across nodes, FWP's ability to provide fairness decreases, but it continues to maintain isolation between protocols so they can operate efficiently. Finally, our results show that the overlapping nature of wireless communication places a fundamental tradeoff between transmit (per-node) fairness and channel (network) fairness.

This paper makes four research contributions. First, it proposes grant to send (GTS) as a mechanism for media access. Second, it presents an example GTS protocol, FWP, and describes how it be the basis of efficient and robust sensornets. Third, it presents solutions to several complexities that arise when applying fair queueing algorithms to protocols on lossy, low-power networks, such as allocation estimate inconsistencies (Section 3.4) and diverse multihop networks (Section 3.6). Fourth, it demonstrates that with these solutions FWP can improve protocol goodput and energy efficiency (Section 4).

Section 5 describes how these contributions draw from the large corpus of prior work. FWP builds on wireless network research on layer 3 fair rate control [36, 12, 24] and layer 2 fair packet scheduling [1, 4] to define a unifying mechanism that sits between the two. As nodes may have differing traffic loads, do not have uniform connectivity, and the network is lossy, FWP cannot use existing fair queueing algorithms verbatim, but must instead adapt and adjust them to work in light of the challenges encountered in low-power wireless networks. Similarly, while these has been a great deal of investigation of how to apply fair queueing to wireless networks [16, 31], these techniques either require being able to control the MAC, which is not always possible, or in the case of sensornets are concerned with node fairness within a protocol rather than fairness between protocols.

Section 6 discusses the results of this paper and their significance in the scope of the larger question of sensor network architectures. Taking advantage of multihop information on traffic patterns and connectivity, network-layer rate control can generate a good estimate of when to send another packet. Deciding when to actually send that packet, however, is up to the MAC layer. FWP demonstrates that a small amount of cross-layer communication — in this case, a GTS quiet time — can improve network performance and robustness while remaining flexible enough to have a broad range of uses.

2. The Fair Waiting Protocol

This section describes the motivation behind the grant-to-send mechanism and how it can be used as the basis for mulithop wireless fair queueing. It describes FWP, a GTS protocol.

2.1 Motivation

Unlike the Internet, which uses a single network (layer 3) protocol, sensornets typically use several protocols concurrently [2], each of which has different traffic patterns and timing expectations. When protocols violate each other's assumptions, they can greatly reduce energy efficiency and cause failures. For example, some routing protocols use rate control to avoid self-interference along a path [36]. Another protocol, however, can easily disrupt this by sending bursts of packets during the desired inter-packet intervals. For multiple network protocols to be able to efficiently coexist, the network must provide mechanisms to avoid inter-protocol interference. As timing requirements are the result of a local computation, but transmission decisions are not limited to a single node (another node could send an interfering burst), the logical place to introduce multi-hop interference avoidance mechanisms is below the network protocols.

Multihop-aware interference avoidance only solves part of the problem. While it causes protocols to respect the transmissions of others, it does not promise that a protocol can transmit. In addition to isolating concurrent packets from each other, the network needs to isolate sources of packets so that each achieves a fair share of

the available bandwidth. In wired networks, fairness algorithms use the time a packet is on the medium, rather than packet counts, in order to not penalize protocols that send many short packets. In this case, however, the wireless medium means that protocols often want to hold the channel but leave it unused to prevent self-interference. These time intervals lend naturally to calculating a protocol's utilization of the channel.

There are challenges in mapping these approaches to a real protocol, however. In a wireless network, these fairness calculations are distributed, rather than centralized: when a node transmits a packet for a protocol, the protocol uses not only the channel of the source, but also everyone who hears it, and this must be taken into account. Furthermore, packet losses mean that this distributed information can become inconsistent.

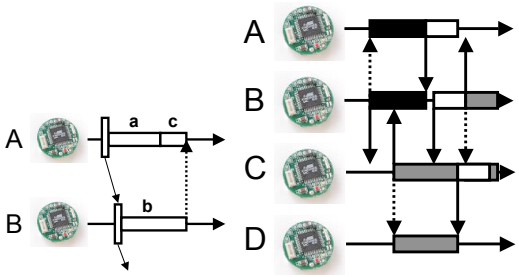
An abstraction that causes protocols to collaboratively share the channel without explicit coordination would be a useful and powerful building block when constructing large and complex applications. Protecting protocols from each other's traffic would minimize unforeseen interactions and isolate failures, leading to more composable and more robust systems.

2.2 GTS and FWP Overview

When a network protocol sends a GTS packet, it may specify a post-transmission quiet time during which only the recipient may transmit. As a transmitter must obey the quiet time, when it transmits it allocates and grants the channel for exclusive use by the destination, which can in turn grant the channel to another node.

Figure 1(b) shows how GTS tabulates quiet time over an example multihop packet exchange. The exchange begins when B sends a packet to C. Both B and A must obey this quiet time (black), while C, being the recipient, does not. C sends a packet back to B; both C and D must obey the quiet time (grey), but B must wait until its black quiet time expires before transmitting again. When the black quiet time expires, A wins CSMA between B and A and sends a packet to B. B forwards the packet to C, who waits until the end of his quiet time before forwarding it to D. When B's quiet time from this exchange expires it send a packet to A.

FWP is a GTS protocol that uses grant durations and packet transmission times to estimate how long each protocol has occupied the channel. Given a set of transmission requests from different protocols, FWP transmits a packet from the protocol that has occupied the channel least. This is a simple version of the basic fair queueing algorithm defined by Demers et al [4]. Unlike traditional fair queueing algorithms, which allocate bandwidth on a single node, GTS allocates the channel access time across a region of the network. In a wired network, links are independent and network protocols move packets from one link to another. In a CSMA-based wireless network, the medium is a continuous space which many nodes share and can influence.



(a) Quiet time overlap.

(b) A complex GTS multihop packet exchange.

Figure 1: How quiet time is calculated, allotted, and used to schedule packets. Solid lines are received packets, dashed lines are overheard packets, and the boxes are post-transmission quiet times.

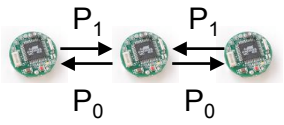


Figure 2: Distributed fairness in multihop networks. For all nodes to observe channel fairness, they must see the channel utilized 50% by P0 and 50% by P1. In multihop networks, obtaining perfect fairness simultaneously can be impossible.

Furthermore, as multiple nodes share the channel and connectivity is not transitive, quiet times can easily overlap. Figure 1(a) illustrates how FWP tabulates channel occupancy when quiet times overlap. If a quiet time interval b overlaps an existing quiet time a, then FWP only considers the packet to allocate the channel for the period which b extends beyond a (the interval c in the figure).

Because FWP allocates time over space and receivers do not adjust their allocations, nodes in single-hop communication range do not agree on how much different protocols have used the channel. Consider the case shown in Figure 2, where there are three nodes, two of which are hidden terminals to each other. If these two nodes transmit packets of protocol P_1 but the center node transmits packets of protocol P_0, it is not possible to have fair channel utilization at every node. Fairness at the edge nodes means each observes P_0 and P_1 using an equal share of the channel. For the center node to observe fairness, the two P_1 transmissions would have to be at exactly the same time, causing the hidden terminal problem and preventing the center node from observing them. The three nodes cannot all observe channel fairness.

Value	Meaning
tx	Transmit time of a packet.
q	Quiet time a packet requests.
d	Duration of a packet: $tx + q$.
T_p	Time that p has occupied the channel.
Q_p	End of p's quiet time.
R_p	Requested quiet time of p's next packet.

Table 1: Notation for values in FWP. p is a protocol, defined by a protocol identifier in a data-link packet header (e.g., AM type).

2.3 FWP Details

Table 1 summarizes the notation used to describe FWP. FWP keeps track of three values for each protocol p in the system. The first is T_p, the amount of time that a protocol p has occupied the channel. The second is Q_p, the end of p's quiet time, that is the earliest time at which FWP may send a packet while respecting p's grants. The third is R_p, the quiet time request of the next packet from protocol p. If FWP has packets from multiple protocols outstanding, it selects the p with the smallest T_p and submits it to the data link layer for transmission when all quiet times Q have expired.

When FWP overhears or transmits a packet of protocol p, it increases T_p and sets Q_p to be the current time plus d. A packet increases the channel allocation of and enforces a quiet time on everyone except the receiver. From the receiver's perspective, the packet has not reserved the channel. As everyone receives broadcasts, they increase T_p and Q_p only on the transmitter.

The example in Figure 1(a) shows how this algorithm leads FWP to handle overlap in quiet time intervals. As mentioned earlier, when a FWP hears a packet from protocol P with a quiet time t_p, it does not automatically add t_p to P's channel allocation T_p. Rather, it adds the amount that t_p extends beyond the current quiet time $(max(Q_p) - t_p)$ FWP measures the time a protocol actually has the channel allocated to itself, not the sum of its reservations. This approach also applies to overlapping between multiple protocols; the first protocol pays for its full interval, but other protocols pay only for how much their interval extends beyond it. In Section 6 we discuss an example of why this overlapping is desirable and how it can enable a class of bandwidth-intensive protocols.

3. Abstract Loads

This section defines the fairness and isolation metrics used in this paper. It evaluates FWP's fairness and isolation in single-hop and multihop networks under abstract protocol loads on small single hop networks and a large multihop testbed. The protocol loads are abstract in that they are inexhaustible sources of packets generated at every node, with no correlation between individual transmissions. Abstract loads allow evaluate FWP in simple circumstances without complex inter- and intra-protocol dynamics. This section defines the fairness and isolation metrics with which we evaluate FWP. The next section evaluates FWP under real protocol loads in a multihop sensor node testbed.

Fairness	Description
Channel	Time different protocols allocate channel at a node (RX or TX).
Transmit	Time allocated at a node by transmissions of different protocols.
Node	Time allocated by different nodes for a single protocol.

Table 2: Three fairness metrics used to evaluate FWP. All metrics use the Jain fairness index (JFI).

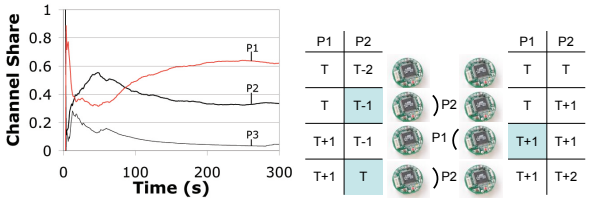
3.1 Fairness and Isolation

Table 2 summarizes the three forms of fairness that are relevant to FWP. The first is *channel fairness*, or how fairly the channel is allocated to different protocols between all of the nodes in a region of the network. The second is *transmit fairness*, or how fairly a given node schedules its own packet transmissions. The third is *node fairness*, or how fairly a protocol's time on the channel is spread across nodes. A network where only one node transmits packets can have high channel and transmit fairness but low node fairness. In the best case, a network provides all three. As sensornets CSMA protocols typically do not use exponential backoff, they give each node an equal chance of acquiring a channel and do not suffer from the backoff capture effect that occurs in exponential schemes [23].

This paper measures fairness using Jain's fairness index (JFI) [14]:

$$f(x_1, x_2, x_3, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$$

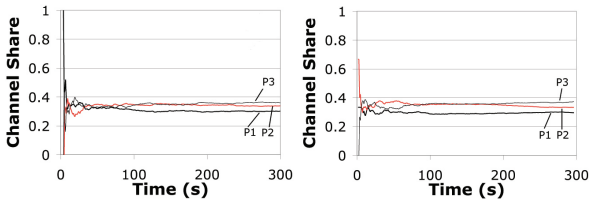
The second metric for FWP is isolation, which measures how well it causes protocols to respect each other's quiet times. We measure isolation in the context of a one-hop network by having a single node be the recipient of all packets. Each transmitter has a fixed number of packets to send after which it stops. If quiet times are observed perfectly, then the time for all nodes to complete their transmissions should be greater than or equal to sum of all d, with a bit of extra time for initial CSMA backoff. We quantify isolation as the fraction of expected time. If $\sum d$ is 400 seconds and the transmissions take 360 seconds, then the experiment had an isolation index of 0.9. If an experiment takes longer than $\sum d$, we give it an isolation of 1.0. This metric assumes that there is no external interference that might cause a large number of backoffs or packet corruptions. All experiments with real motes used 802.15.4 channel 26, which experiences minimal interference from external sources such as 802.11b [29].



(a) Protocol channel shares over time. This node has a transmit fairness of 0.66.

(b) Ping-pong effect.

Figure 3: Packet loss can lead to low transmit fairness. The plot shows the per-protocol outgoing bandwidth for one node in a 5-node network. The cause is the ping-pong effect, shown on the right.



(a) Decaying T

(b) Flushing T

Figure 4: Effect of decaying and flushing T periodically to filter out loss-based inconsistencies. For the node shown, decaying every second leads to a transmit fairness of 0.994 and flushing every second leads to a transmit fairness of 0.991.

3.2 Uniform Lossless Load

In an ideal network, where every node offers the same load and there are no collisions and no packet losses, FWP provides channel, transmit, and protocol fairness. Because every node sees the same packets, every node maintains an identical estimation of T for each protocol. Therefore, they all agree which protocol should be sent next and the next node to transmit will choose the one with the smallest T. This provides channel fairness. As each node has an equal chance of being the transmitter when a quiet time expires, each node transmits an equal share of each protocol's packets, providing both transmit and protocol fairness. Because no packets are lost or collide, there is perfect isolation: every node waits until the current quiet time expires, then one transmits.

3.3 Isolation and CSMA

CSMA is inherently imperfect. First, it is possible that two nodes sample the channel at the same time and both decide to transmit. Second, radios take time to transition from receive (channel sensing) to transmission mode. During this RX/TX transition, another node could sense a clear channel and decide to transmit. We measured FWP's isolation on a small number of nodes in easy radio range of each other, and discovered that even in single-hop network its isolation was 0.95.

Examining the traces, we found the cause to be the RX/TX transition time. The standard TinyOS stack chooses initial backoff periods in the range of 0.3-10ms, but only chooses from 32 possible values. It chooses between 10 and 320 "jiffies" (ticks of a 32kHz clock) with a 10 jiffy granularity. Because the expiration of a quiet time synchronizes the CSMA transmission requests of multiple nodes, then for n nodes the chances that two nodes collide on the earliest slot is $(1 - \frac{31}{32}^{n-1})$. A collision will cause at least three packets to be transmitted in one quiet period: the first two that collide, followed by a node that picked a later backoff value.

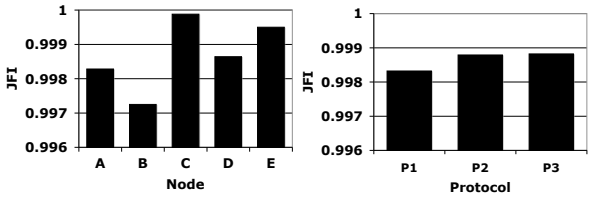
We explored several solutions this problem, such as adding jitter to quiet times and introducing reception latencies. The best solution turned out to be to simply remove the unnecessary quantization of the CSMA backoff. Rather than choose one of 32 backoff values with a 10 jiffy granularity, the stack chooses values with a granularity of 1 jiffy. This restores FWP's isolation to 1.0.

3.4 Uniform Load with Loss

The simple version of FWP described above is not without faults. For example, a protocol can accumulate a large T and later be starved by protocols who have been quiet for a long time. A second and greater issue is that packet losses can greatly complicate fair channel allocation. Even in a single hop network, lost packets cause nodes to have inconsistent views of T. The nodes disagree on how to achieve channel fairness, leading to low transmit fairness values.

Figure 3 shows the normalized packet transmissions of one of five Telos nodes in a single hop network. Each node has an abstract load of three protocols with quiet times of 20, 40, and 80ms. The network has high channel fairness (0.99). While protocols are receiving a fair share of the channel, nodes are servicing protocols unevenly. For example, the transmissions of the node shown in Figure 3 allocate the channel with a ratio of 8:15:1 rather than 1:1:1, leading to a transmit fairness of 0.66.

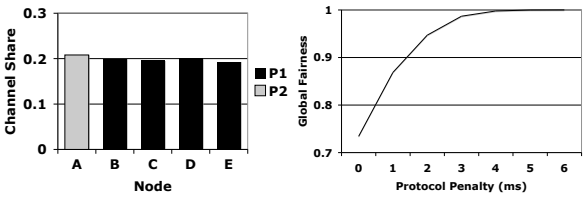
This behavior is due to packet loss. If one node drops a packet of protocol P_0, then its perception of channel utilization will differ from that of nearby nodes. Its estimation of T_0 will mean that it thinks P_0 should be transmitted next. If it transmits P_0, however, all other nodes will think P_0 has used the channel significantly more, and so only transmit other protocols. Nodes have differing opinions on which protocol needs more time on the channel, and do not allocate their transmissions fairly. While the overall network allocation is reasonably fair, the per-node allocation is not. This can lead to local star-



(a) Transmit fairness at nodes A-E.

(b) Node fairness for protocols P1, P2, and P3.

Figure 5: Fairness in a single hop lossy network for five nodes offering uniform load of three protocols with quiet times of 20, 40, and 80ms and periodically decaying T. The channel fairness is 0.9995.



(a) With no protocol penalty, channel fairness is 0.746.

(b) Channel fairness for different protocol penalties.

Figure 6: Packet transmissions with varied load. Node A sends only P1 and nodes B-E send only P2. A protocol penalty restores the channel fairness from 0.746 to 0.9978 with a 4ms penalty, 0.9998 with 5ms and 0.9999 with 6ms.

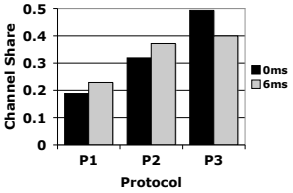


Figure 7: Fairness under complex varied load. One node sends P1, two send P2, and three send P3, all with a quiet time of 25ms. A 6ms protocol penalty improves channel fairness from 0.876 to 0.951.

vation.

As FWP's selection of transmissions is deterministic, inconsistencies lead to unfair utilization. We explored three possible several to resolve these inconsistencies: periodically advertising measured channel utilization, periodically *decaying* T by dividing all T_p by 2, and periodically *flushing* all T_p to be zero. Periodically advertising channel utilization could give nodes consistent views. However, as Figure 2 showed, these values inherently vary over space; periodic advertisements lead to unfair allocation within the network, as local variations that could be overcome are overwritten instead. Furthermore, in multihop network channel allocation inherently varies across nodes, and so resolving to a single uniform value does not reflect local channel fairness.

Figure 4 shows the effect of periodically halving or flushing T. Both approaches restore transmit fairness. But how often should FWP decay or flush T? We address this question in the context of a multihop network in Section 3.6. As decaying T changes which protocols are transmitted but not when, it has no significant effect on isolation.

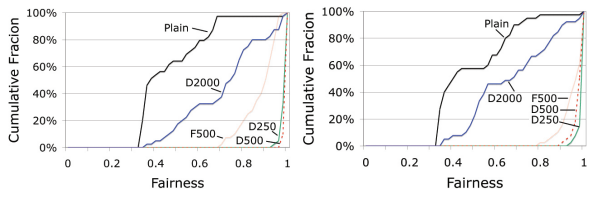
3.5 Varied Load

The previous experiments explored FWP's fairness and isolation when all nodes in a single-hop network offer uniform load. When nodes have varied loads (the set of protocols which have pending packets), FWP must deal with the fact that the underlying CSMA layer arbitrates access to the channel. Figure 6(a) shows the distribution of transmissions for a single-hop network of Telos nodes offering varied load. One node sends packets of protocol P_1 while four nodes send packets of protocol P_2. Both protocols have the same quiet period. Fair channel allocation would call for the nodes to collectively send an equal number of P_1 and P_2 packets, but the ratio is instead 4:1. This ratio comes from the MAC layer. When a quiet period ends, each node tries to acquire the channel, and there is an 80% chance the node that does so will transmit P_1. CSMA randomly selects a node, and this leads to low global fairness.

To address this problem, FWP institutes a penalty to a protocol whose quiet-time just expired. It does this by increasing Q of the protocol with the lowest T by a small amount. If Q_p is far in the past, then the protocol can be sent normally. However, if it was p's quiet time whose expiration allowed transmission, then other protocols will be able to transmit first, as they only wait until Q_p. Figure 6(b) shows the effect on fairness as this penalty value is increased in the case shown in Figure 6(a). At 6ms, the channel fairness is 0.9999; FWP uses this as its protocol penalty. 6ms is approximately two packet times for the Telos' CC2420 radio.

While the protocol penalty improves channel fairness in this simple case, it has several limitations. First, if all nodes agree on channel utilization and still wish to transmit the penalized protocol, FWP forces them to wait unnecessarily, reducing network capacity. Second, while it works in the two-protocol case, it degrades as the load becomes more complex with increasing numbers of nodes and protocols. Figure 7 shows the effect in a more complex situation.

Solving this problem perfectly requires global knowledge, but FWP is intended to be a lightweight and local



(a) 20, 40, and 80ms (b) 20, 60, and 140ms

Figure 8: CDF of transmit fairness on the motelab testbed under two abstract packet loads for decaying (D) and flushing (F) T. Decaying T values every 250 or 500ms leads to the best fairness distribution. D500 has a channel fairness of 0.995 in (a) and 0.994 in (b).

mechanism. While FWP does not provide perfect fairness under varied load, its fairness is greater than relying purely on CSMA. The techniques proposed by Vaidya et al. to provide MAC-level fairness [31] for wireless flows are a fruitful place to start on investigating how to improve FWP's fairness under varied load; we plan to explore such possibilities as future work.

3.6 Multihop Fairness

Section 3.4 showed that inconsistencies between T estimates can harm transmit fairness, and that in a single-hop network periodically flushing or decaying T values restores it. A multihop network poses a more difficult situation, as its traffic loads may be inherently unfair. Flushing and decaying were effective, the rate at which FWP performs them can have a significant effect. Flushing every packet time would clearly be problematic. The update intervals may depend on the quiet times specified: decaying quiet times of 10ms every 250ms may be feasible, but decaying times of 250ms at the same rate might reduce fairness.

To explore these issues and get a sense of how often FWP should decay T, we ran two abstract loads on Motelab. One load has nodes send packets with quiet times of 20, 40, and 80ms. The other has quiet times of 20, 60, and 140ms. Our belief was that an interval along the lines of the least common multiple would be desirable, and chose the second load such that it would be comparatively large (420ms rather than 80ms). We evaluated the flush and decay policies at intervals of 125, 250, 500, 1000, and 2000ms as well as with no T updates by examining the distribution of transmit fairness values.

Figure 8 shows a subset of the results. Large intervals are disastrous as is using T without periodic adjustments. For both workloads decaying T every 250 or 500ms works best, with each one being slightly superior in a different workload. We are uncertain why these values are the most effective. Given all of the factors involved (topology, loss, CSMA, load), we consider a further investigation of this topic an important area of future work beyond the scope of this paper.

Event	Action
Hear or transmit protocol p	Increase T_p and Q_p by d
Decay interval expires (500ms)	Halve all T
Quiet times expire If p was last protocol heard expiration time += protocol penalty	Submit p of $min(T)$ to CSMA

Table 3: The complete FWP protocol. The notations are from Table 1.

4. Real Loads

The prior section described FWP and evaluated how it performs under abstract protocol loads. This section summarizes the complete FWP algorithm and evaluates it on Motelab under real traffic loads generated from existing sensornet multihop protocols.

4.1 Complete FWP

The complete FWP is the version presented in Section 3.4 with periodic 500ms T decay and a protocol penalty of 6ms. Table 3 summarizes the protocol. Implemented in the TinyOS 2.0 operating system as an alternative queue implementation between network protocols and the data link layer, FWP is approximately 270 lines of nesC code. It requires two bytes of additional state (the end of the longest quiet time) plus three bytes per network protocol. Two bytes store the T_p value and one byte is used to translate between key namespaces in the generic components (there may be multiple senders of a single protocol, but each protocol should still only get one share).

Just as in TinyOS 2.0, a FWP data link layer provides each instance of the packet sending service AMSenderC with a queue of depth one. Protocols may place their own queues on top of this in order to have greater buffering. These queues isolate network protocols from one another in terms of buffer management, such that one protocol cannot monopolize the queue. The standard TinyOS implementation services senders with a round-robin policy. FWP merely changes this to service them based on T values. The implementation keeps time in millisecond granularity, and allows protocols to specify a d of 0-255ms (up to ≈75 packet times for the commonly used data link layer in sensornets today, 802.15.4), adding a single byte header to all packets.

Properly implementing FWP requires being able to cancel a packet once it has been submitted to the data link layer. We modified the standard CC2420 radio stack to support this functionality. The TinyOS packet sending interfaces have a `cancel` function, but this is only implemented at the queueing level. Once the radio has begun CSMA to transmit, network protocols cannot cancel

the transmission. As FWP's quiet time synchronization causes many nodes to enter CSMA at the same time, we added the ability to cancel into the data-link stack. When the data link layer cancels a packet in the FWP stack, this stops the CSMA backoff timer, flushes the packet from the radio packet buffer, and returns to a listening state.

Because FWP requires being able to overhear packets and the CC2420 radio does not support hardware-generated acknowledgements unless there is address filtering, the FWP stack uses software-level acknowledgments. We are currently discussing with ChipCon possible changes in next generation radios that might let them better support FWP and other sensornet protocols.

4.2 Real Traffic Loads

All of the previous experiments used abstract packet loads of protocols that are not truly multihop. As FWP actively silences transmissions over possibly dense areas of a network, it could significantly decrease network capacity. While interference avoidance may improve network goodput, more retransmissions may be just as effective. This raises the question: how does FWP affect network goodput and efficiency?

To evaluate whether FWP meets its goals under real traffic loads, we implemented three multihop protocols from the literature: a rate-controlled tree collection protocol proposed by Woo et al. [36] (ARC) and two dissemination protocols, PSFQ [33] and Trickle [15]. We implemented ARC so that we could run multiple instances as separate protocols to capture more complex dynamics.

4.2.1 Network Protocols

ARC assumes an application produces traffic at a constant rate. It uses a probabilistic scheme to meter this rate with a dynamically estimated probability p. When ARC receives a request to send a packet, it sends the packet with probability p. When the network is underutilized, p is 1, and it decreases as contention increases. ARC adjusts p using additive increase multiplicative decrease based on whether the next hop successfully forwards a packet and observes this through packet snooping. To separate out bad route selection from data interference, the ARC implementation uses a static routing tree which we derived from connectivity experiments. A quiet time in ARC allows a transmitter to hear the next hop forward the packet without interference. Our ARC implementation has a five packet queue.

PSFQ is a reliable dissemination protocol that uses NACKs to locally recover from lost packets. A source generates data items, which the network propagates using a controlled flood. Every data item has a sequence number, and nodes maintain cache of recently heard items. If a node detects it has missed a sequence number, it sends a NACK and neighbors respond with the missing item. These responses have random jitter and suppress each other in order to prevent a response implosion. A quiet time in PSFQ forces a data transmitter to wait to hear for a possible NACK.

Protocol	R_p	**Interval**
ARC root	0ms	0
ARC non-root	10ms	1s
PSFQ	10ms	1s
Trickle (τ_l =1s, τ_h =1024s)	0ms	60s

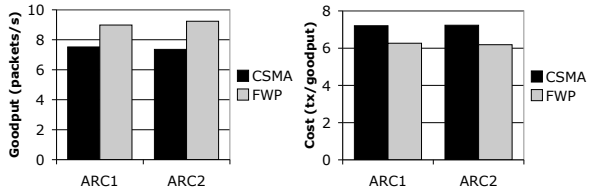
Table 4: Configuration of network protocols for multihop tests. R_p is the requested quiet time of a packet, and interval is the time between separate packet generation events. For ARC this is nodes sending to the root, for PSFQ and Trickle this is the root disseminating to the nodes.

Trickle is a reliable dissemination protocol that dynamically adjusts its transmission intervals in order to trade off between increasing propagation speed and decreasing the cost of detecting missed items. Every node advertises its data at a random point in an interval of length τ, but suppresses its transmission if it hears another node advertise the same data. Because Trickle uses network-level rate control, it does not institute a quiet time. However, its transmissions obey the quiet times of others. We used a modified version of the standard dissemination protocol implementation (`DisseminatorC`) in TinyOS 2.0.

4.2.2 Experimental Setup

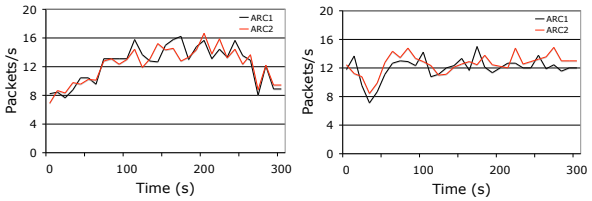
The following experiments all ran on the Motelab sensornet testbed at Harvard University [35]. This testbed consists of 180 Telos nodes attached to the ceiling of a busy office environment with many obstructions. We used 40 of the 180 nodes. As the nodes are on the ceiling, there are few changes in line-of-sight or obstacles that vary over time: the received signal strength between a node pair is generally very stable. However, as the network is in an active office environment, there is a great deal of 802.11b interference with high temporal variations. As 802.11b can interfere with the 802.15.4 radios on micaZ motes, this traffic introduces many complexities to packet delivery rates [29]. We ran all experiments on 802.15.4 channel 25 with a transmit power of -0dBm. Based on experiments running the protocols in isolation and their logic, we assigned q values as shown in Table 4.

Each multi-protocol experiment has two separate ARC trees, over which every node tries to send packets at a rate whose aggregate traffic will use a significant portion of the network capacity without saturating it (1Hz). Pushing packets faster than this leads to the nodes close to the root ignoring forwarding traffic and monopolizing the bandwidth to the root. PSFQ pushes a new data values into the network every second, and a new Trickle value



(a) ARC goodput. (b) ARC cost.

Figure 9: Goodput and cost for two separate ARC instances running in the presence of PSFQ and Trickle on top of CSMA and FWP. FWP increases goodput by 23-30% and decreases cost by 5-10%.



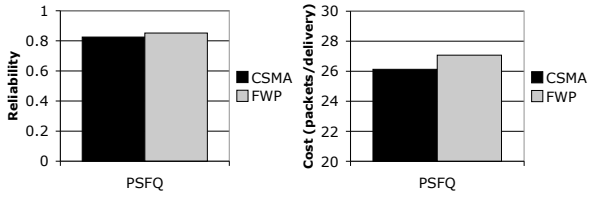
(a) CSMA (b) FWP

Figure 10: ARC goodput over time using CSMA and FWP. CSMA moves between 8 and 16 pps, while FWP maintains a more steady rate around 12pps.

is inserted every minute. We performed this experiment with the standard TinyOS networking stack and with our FWP implementation.

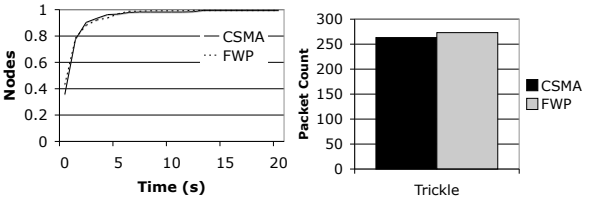
We ran the suite of protocols for ten minutes and collected data on the network behavior. For both the standard TinyOS networking stack and the FWP stack, we measured the goodput at the root of each of the ARC trees, the successful delivery rate of PSFQ, trickle propagation latency, and how long each protocol's packets spent in the data-link transmission queue.

4.2.3 Multi-protocol Results



(a) PSFQ delivery success rate. (b) PSFQ cost.

Figure 11: Delivery success rate and cost of PSFQ on top of standard CSMA and FWP. FWP increases the delivery success slightly (3%) with a an accompanying increase in cost (3%).



(a) Trickle delivery latency. (b) Trickle cost.

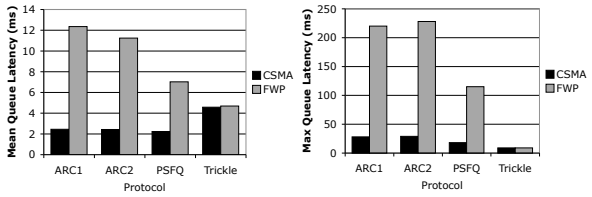
Figure 12: Trickle propagation rate using standard CSMA and FWP. The distribution of delivery latencies is for all nodes in ten separate trickle events (400 data points).

Figure 9 shows how ARC performs. For the configuration in Table 4, using FWP increases the goodput of ARC by 23-30% and reduces the cost per goodput packet by 5-10%. We explored a range of quiet times and transmission intervals. At very low rates, CSMA and FWP performed similarly. At very high rates (e.g., inter-packet intervals of 10ms, where the network generates 4000 packet/s), CSMA outperforms FWP. This occurs because the few nodes close to the base station are completely overloaded and stop routing packets; just send as quickly as they can. In this circumstance, a quiet time of 6ms merely reduces network capacity.

Figure 10 shows ARC's goodput over a five minute interval. Using CSMA, ARC swings between 8 and 16 packets per second. Using FWP, its rate is more regulated, and except for an early dip in rate it stays close to 12 packets per second. Using FWP, ARC's rate control is more stable. 12 packets per second, however, is well below network capacity.

The effect of ARC's rate control on other protocols can be see in Figure 11, which shows the results for PSFQ. Using FWP increases the reliability of PSFQ by 3% at a 3% increase in cost. By applying rate control, ARC is not saturating the channel, and so PSFQ can perform reasonably well. FWP allows PSFQ to catch a few extra stragglers. Figure 12 shows the results for Trickle. Under CSMA and FWP Trickle behaves almost identically: FWP sends 5% more packets.

Figure 13 shows the queue latency distributions for ARC, Trickle, and PSFQ. Quiet times introduce much higher queue latencies for ARC and PSFQ, which are sending approximately 50 and 20 packets per second, respectively. In contrast, Trickle, which sends approximately 20 packets per minute, has FWP queue latencies equivalent to those of CSMA. These queue latencies can also explain why PSFQ sends more packets; they are larger than PSFQ's random jitter in responses to NACK packets, so the chances that two nodes respond at the



(a) Mean latency. (b) Maximum latency.

Figure 13: Queue latencies for the four protocols. Because they use the channel most, ARC packets have to wait longest in queues. The average and worst case queue latencies of Trickle packets, in contrast, are within 100μs of each other. FWP favors protocols which use less than their fair share.

same time goes up. Even though ARC has higher queue latencies (the maximum is 28ms for CSMA and 228ms for FWP), FWP improves both goodput and efficiency.

These improvements are smaller than we had hoped for: the large amount of work in fair rate control [5, 24, 36] suggests that self-interference and inter-protocol interference are significant problems in sensornets. However, considering the protocols we used, the reason why FWP did not meet our expectations is obvious with hindsight: these are well designed protocols that take active measures to avoid these problems using through rate control and suppression. ARC, for example, adjusts its utilization of the channel to be low enough that the network does not approach saturation, unless the load is so high that it devolves into a single-hop network. Although the network is generated 40 packets/s, the base stations receives 7-9 packets per second.

5. Related Work

While fair queueing algorithms are generally ambivalent on what division they provide fairness over, in Internet systems it is typically an IP flow (node/node/port) or just a node pair. In wired systems, fair queueing is a local problem: a node has a dedicated channel that it can use as it sees fit. The medium inherently isolates separate nodes from interfering. In a landmark paper, Demers et al. showed a system can fairly queue in the wired case with the very simple algorithm [4], which FWP adapts to the low-power wireless domain. For high-capacity routers, per-packet processing costs are a significant concern, leading to a series of subsequent algorithms [28, 30] that trade off precision for efficiency. In sensornets, the processing/communication tradeoff is reversed, as CPUs are mostly idle [27], simplifying the problem and making accurate algorithms feasible.

One challenge wireless networks pose is that several nodes share the channel being allocated, and this relationship is neither binary nor symmetric. As each node may be servicing a different number of flows, the simple MAC-level node fairness can be a liability. Vaidya et al. showed one way to resolve this tension in a distributed fashion, making CSMA backoff inversely proportional to a flow's channel use [31]. As the authors note, that the fairness of this approach degrades with packet loss, and it is designed for a single- rather than multi-hop network. When many nodes participate in the fairness unit (e.g., a protocol), then the shared channel allocation state is further distributed.

MAC layers encounter channel allocation challenges as a matter of course. In RTS/CTS, for example, a node requests permission to transmit. For traffic loads of large bursts of traffic, this control overhead can be amortized over many data packets. For small bursts, as is often the case in sensornets, a hidden terminal in the RTS/CTS are just as pronounced, limiting its utility. Furthermore, as noted by MACAW, RTS/CTS has edge cases that lead to inefficient use of the channel [1]. These issues have prevented SMAC [39], although the first MAC designed specifically for sensornets, to have limited use. Instead, systems commonly use BMAC, a simple CSMA MAC layer with optional low-power listening [21].

In the context of sensor networks, one multihop protocol that fits well into a node-based or flow-based accounting approach is collection [38]. Collection protocols deliver data from every node in a network to one or more collection roots or base stations. Because nodes form a tree, shallow nodes can receive an unfairly large share of the endpoint bandwidth. Algorithms to provide fairness in these circumstances [5, 24, 36] are complimentary to this paper; they provide inter-node fairness within a protocol, and benefit from a lower layer that provides inter-protocol fairness across nodes.

The FPS protocol and its successor, Twinkle, use explicit slot allocation to schedule flows up a collection tree [10]. Using explicit slot allocation allows FPS to greatly reduce energy consumption, as nodes can remain asleep for unused slots. Additionally, as it allocates entire flows from source to sink, can control how much of its incoming bandwidth is given to each child and provide network-level fairness. FPS demonstrates that a dedicated network with a single network protocol can be heavily optimized through explicit reservation. In contrast, FWP takes an ad-hoc and opportunistic approach, allocating the channel as it is needed. The distinction between the two approaches is akin those between virtual circuit and wormhole routing [3] in parallel architectures or virtual circuits and packet switching in IP networks, with similar tradeoffs between state, control overhead, and flexibility.

As sensornet applications grow in complexity, so does

the need for effective integration. The SP programming abstraction [22] proposes two software mechanisms, a send pool and a neighbor table, to decouple network protocol implementations while simultaneously allowing an OS to couple their traffic. The goal of SP is to maximize how efficiently a node OS can use its local resources, e.g., by sending rapid bursts of packets. Decoupling network protocols so completely from transmission scheduling, however, precludes certain effective mechanism such as network-level rate control. SP and FWP seek the same goal, but take opposite approaches: SP optimizes below and leaves fairness to the layers above, while FWP provides fairness below and leaves optimizations to the layers above.

6. Discussion and Conclusion

In dense networks where there is heavy traffic, FWP improves network goodput and efficiency while enabling multiple protocols to fairly and collaboratively share the channel. However, these benefits do not come without costs. In sparse networks, where interference is less of an issue, FWP's quiet periods and protocol penalties can reduce network capacity. This is not surprising; in a network with a single active protocol that needs to optimize use of the channel, isolation mechanisms are unnecessary overhead.

The experiments in this paper used constant quiet times derived from isolated traffic or protocol logic. These values are dependent on the radio hardware (packet times), environment, and network topology. It therefore seems likely that robust protocols built on top of FWP would need to dynamically estimate this parameter, just as some estimate transmission rates.

Grant-to-send is a significant departure from existing approaches. In adversarial situations, or networks where nodes have differing goals, this mechanism is counterintuitive. In a sensornet, however, where nodes are trying to achieve a common goal (and are under a single administrative domain), more altruistic approaches are possible. It can be used to implement several forms of network-scheduling. Section 4 showed how ARC could use it express rate control to other protocols and how PSFQ could use it to reserve its local channel for NACK responses. A quiet time could also be used for bulk unicast data transfer: the requester sends a large grant to the provider, who can then send a large number of data packets with quiet time 0.

FWP poses interesting questions for routing protocols. If trying to optimize goodput, a protocol routing along a path will try to control how far its quiet time reaches. A quiet time that reaches two hops away and prevents rapid forwarding is undesirable. Encoding flows – problematic in sensornets, where not all traffic patterns are flows – to control the set of nodes who do not have to obey a quiet time could allow network protocols to make more flexible routing decisions.

FWP takes no position on conserving energy. It assumes that nodes can snoop on packets, but in the common case of very low traffic rates, it does not preclude energy management techniques. However, as important as energy is in sensornets, we believe that in this case it should be the second, not the first, step. Successful systems and networking specifications such as files, threads, and TCP have shown us that simple designs are flexible, and flexible designs can be highly optimized later. Many of the current approaches for network energy management, such as communication scheduling [10, 17], are only feasible in isolation as they assume complete control of the system. Therefore while they guide research and our understanding towards better approaches, they are difficult or impossible to apply in the general case, especially in combination.

The tremendous challenges in developing sensornet applications, however — networking, systems, and otherwise — have led to many system architectures with the goal of development. SP [22] and the modular network layer [6] propose programming abstractions. Tenet [9] and EmStar [7] specify computational responsibilities. These architectures have made developing applications easier, but those applications are still vertically integrated systems. We believe that sensornets also need an orthogonal effort to develop a *network* architecture which defines how nodes communicate and the division of protocol responsibilities. The basis of such an architecture is a "narrow waist" protocol, independent of any particular hardware or software platform, that allows multiple multihop layers to operate independently but efficiently share the channel [2]. We believe FWP is a significant step towards this goal.

Acknowledgements

We would like to thank Rodrigo Fonseca and Sukun Kim, whose rate control research provided the inspiration for FWP. We would like to thank Intel Research and Harvard University for providing networking testbeds to the research community. This work was supported by generous gifts from the Intel Corporation and Docomo Capital, a scholarship from the Samsung Lee Kun Hee Scholarship Foundation, the National Science Foundation under grant #0615308 ("CSR-EHS"), and a Stanford Terman Fellowship.

7. REFERENCES

[1] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: Media access protocol for wireless lans. In *In Proceedings of the ACM SIGCOMM Conference*, 1994.

[2] D. Culler, P. Dutta, C. T. Ee, R. F. andJonathan Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a sensor network architecture: Lowering the waistline. In *Proceedings of the Tenth Workshop on Hot Topics in Operating Systems (HotOS-X)*, 2005.

[3] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[4] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, pages 1–12, New York, NY, USA, 1989. ACM Press.

[5] C. T. Ee and R. Bajcsy. Congestion control and fairness for many-to-one routing in sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 148–161, New York, NY, USA, 2004. ACM Press.

[6] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica. A modular network layer for sensornets. In *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[7] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*, pages 201–213, New York, NY, USA, 2004. ACM Press.

[8] O. Gnawali. Private communication, 2006.

[9] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The TENET architecture for tiered sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (Sensys)*, 2006.

[10] B. Hohlt, L. Doherty, and E. Brewer. Flexible power scheduling for sensor networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, Berkeley, CA, Apr. 2004.

[11] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, New York, NY, USA, 2004. ACM Press.

[12] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 134–147, New York, NY, USA, 2004. ACM Press.

[13] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.

[14] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report TR-301, DEC Research, 1984.

[15] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks. In *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[16] S. Lu, V. Bharghavan, and R. Srikant. Fair scheduling in wireless packet networks. In *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 63–74, New York, NY, USA, 1997. ACM Press.

[17] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *Transactions on Database Systems (TODS)*, 2005.

[18] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI)*, Dec. 2002.

[19] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM Press.

[20] J. Newsome and D. Song. Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 76–88. ACM Press, 2003.

[21] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the Second ACM Conferences on Embedded Networked Sensor Systems (SenSys)*, 2004.

[22] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, New York, NY, USA, 2005. ACM Press.

[23] K. K. Ramakrishnan and H. Yang. The ethernet capture effect: Analysis and solution. In *Proceedings of the IEEE 19th Local Computer Networks Conference*, Oct. 1994.

[24] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 63–74, New York, NY, USA, 2006. ACM Press.

[25] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: a hybrid mac for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 90–101, New York, NY, USA, 2005. ACM Press.

[26] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler. Design and implementation of a sensor network system for vehicle tracking and autonomous interception. In *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN)*, 2005.

[27] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 188–200, New York, NY, USA, 2004. ACM Press.

[28] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 231–242, New York, NY, USA, 1995. ACM Press.

[29] K. Srinivasan, P. Dutta, , A. Tavakoli, and P. Levis. Understanding the causes of packet delivery success and failure in dense wireless sensor networks. Technical Report SING-06-00, 2006.

[30] S. Suri, G. Varghese, and G. Chandranmenon. Leap forward virtual clock: a new fair queuing scheme with guaranteed delays and throughput fairness. In *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, page 281, New York, NY, USA, 1997. ACM Press.

[31] N. H. Vaidya, P. Bahl, and S. Gupta. Distributed fair scheduling in a wireless lan. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 167–178, New York, NY, USA, 2000. ACM Press.

[32] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, Nov. 2003.

[33] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: a reliable transport protocol for wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sens or networks and applications*, pages 1–11. ACM Press, 2002.

[34] L. Wang. MNP: multihop network reprogramming service for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 285–286, New York, NY, USA, 2004. ACM Press.

[35] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: a wireless sensor network testbed. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 68, Piscataway, NJ, USA, 2005. IEEE Press.

[36] A. Woo and D. E. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the seventh annual international conference on Mobile computing and networking*, Rome, Italy, July 2001.

[37] A. Woo and T. Tong. Tinyos mintroute collection protocol. tinyos-1.x/lib/MintRoute.

[38] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of multihop routing in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, Nov. 2003.

[39] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *In Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York, NY, June 2002.