# Bandwidth Management in Wireless Sensor Networks

Bret Hull, Kyle Jamieson, and Hari Balakrishnan
MIT Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
{bwhull, jamieson, hari}@lcs.mit.edu

April 8, 2003

## Abstract

Wireless sensor networks are often used in monitoring and control applications, where software running on general-purpose computers "pull" information from remote sensors and "push" actuations into the network. The sensors themselves form a multihop wireless network communicating with one or more *sensor access points* (SAPs) that interface between application software and the sensor network. This paper addresses the problem of managing wireless network bandwidth and improving network capacity in a sensor network deployed as a *shared infrastructure*, concurrently used by different applications.

Our bandwidth management architecture incorporates three ideas: first, we develop a simple *rule system* that allows applications and the network administrator to specify how traffic generated by sensors should be treated by the sensor network. Each rule is a function that maps a sensor data type and generated value to a transmission rate and a traffic class. Second, we show how using multiple SAPs and *SAP selection method* that considers packet loss probabilities, path load, and path lengths improves the capacity of the network and the performance of individual sensor streams. Third, we show that *hop-by-hop flow control*, rather than end-to-end congestion control, is a better way to cope with the nature of sensor network traffic and avoids unnecessary packet losses that waste valuable wireless network bandwidth. Our experimental results from a 40-node indoor wireless sensor testbed show that these three techniques are simple to implement and allow scarce network bandwidth to be used efficiently.

## 1 Introduction

Wireless sensor networks are often used in remote monitoring and control applications. Two resources—wireless link bandwidth and node energy—are scarce in many sensor networks, and need to be managed carefully. This paper addresses three important problems caused by constrained wireless link bandwidth in sensor networks:

1. How to allocate network bandwidth to sensor streams?
2. How to control network congestion?
3. How to improve the data forwarding capacity of the network?

As sensor nodes get even smaller, and sensor networks grow larger in size, we believe that these bandwidth considerations will become increasingly more imperative and important.

The context for our work is a centrally administered, shared sensor network infrastructure, used concurrently by different applications. As an example, we apply our techniques to an indoor sensor network deployed on one floor of our building, with light, temperature, and acoustic sensors monitoring conditions. This network is concurrently used for different applications, including one that monitors temperature in office and machine rooms, one that monitors light usage, and one that combines these feeds with acoustic data to monitor room occupancy on the floor. In such sensor applications, the relative importance of sensor data streams often depends on the type and values of the data being sensed, and on how data from different sensor streams correlate with each other. For example, if the goal of the temperature monitoring application is to actuate heating or cooling only in those rooms that are occupied, then it would make sense to allocate more network bandwidth for data streams coming from occupied rooms compared to empty rooms. As a more extreme example, if sensors in an area detect abnormally high temperature, it may signify a disastrous event like a fire, in which case it would be prudent to allocate almost all of the bandwidth to those streams.

Shared sensor network infrastructures therefore require a *bandwidth allocation* method, by which the nodes can decide how to allocate network bandwidth to sensor streams. The allocation method has to handle traffic that exhibits a high degree of spatial correlation, when a group of nodes in close proximity all detect an event of interest. It has to be able to change bandwidth allocations in the network depending on observed phenomena.

Our solution to this problem is based on *rules* that allow each node to map any sensor data stream to a desired traffic rate and traffic class depending on its values and other attributed (*e.g.*, the location of the sensed data). The network supports a small number of traffic classes, which the nodes schedule according to priority order. We show how our rule system allows interesting bandwidth allocation decisions to be made by both application developers and the network administrator.

Any network with constrained bandwidth requires *congestion control* to achieve good performance and avoid catastrophic congestion collapse when the offered load exceeds available network capacity along its paths. We show that traditional Internet-style end-to-end congestion control protocols are not well-suited to our domain, and develop a *hop-by-hop flow control* scheme that achieves good performance with low congestion-triggered packet loss rates. This hop-by-hop method allows sources to adapt their transmissions to feedback obtained from nodes further along the path to the destination, without causing high drop rates or entailing the overhead of end-to-end acknowledgments and the slower response of end-to-end congestion control.

Many sensor network architectures, including ours, partition the application. A portion of the application, typically involving sensor data sampling, together with simple filtering, aggregation, and compression, runs on the nodes in the sensor network, with the results of these operations streaming to the portion of the application running on general-purpose computers. This architecture requires a sensor access point (SAP) to interface between the application partitions. Sensor network nodes form a multihop wireless network to communicate data to the SAP.

The problem with a single SAP is that it restricts the capacity of the network to the wireless link bandwidth of the SAP. An obvious solution to this problem is to deploy multiple SAPs to increase the available capacity of the network. If each nodes is able to send data to any one of multiple SAPs, then the overall capacity of the network can be considerably higher than with just one SAP. However, it is not clear how nodes can pick the right SAP to avoid hot spots and realize this potentially higher capacity. We analyze this problem and develop a new routing metric that combines hop-count, path load, and wireless path loss rate for nodes to use while selecting the best SAP. We also show how local rules at a node alleviating traffic oscillations, a common problem in networks where the routing protocol reacts to traffic load.

We have implemented the three components of our bandwidth management solution: a rule system together with priority queuing, a hop-by-hop flow control scheme, and a routing protocol based on load, congestion loss, and hop-count for SAP selection, on TinyOS. In addition to experiments on smaller networks, we present the results of several experiments on a deployed 40-node in-building sensor network.

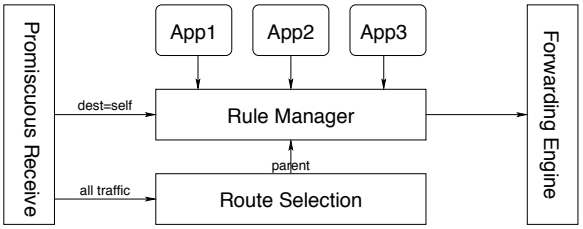


Figure 1: stack

## 2 Design

This section starts by describing the basic application model and network assumptions underlying our work. It then describes the details of the rule system, hop-by-hop flow control scheme, and SAP selection method.

### 2.1 Application model

The data acquisition model for wireless sensor networks depends on the application at hand. While many of the ideas presented in this paper apply to many classes of sensor networks, some are designed with a specific usage pattern in mind. What follows is a description of the type of sensor network applications that motivates our work.

In our framework, an application is simply defined as the logical segment of code running on each sensor node, that acquires the sensor's readings, processes these readings (perhaps correlating them with other sensed values, aggregating values, or compressing them), and then makes a decision as to whether and when it is appropriate to transmit the resulting data. Among applications that follow this model, two broad categories emerge: sensor applications that periodically send readings, and those that send readings contingent on events occurring.

The fundamental characteristic of the first type of application is that the values produced by the application are sent out at a relatively constant rate. The basic structure of such an application is shown in the left-hand column of Figure 2. Periodically, a timer fires, causing a sensor value to be obtained and transmitted over the network. Depending on the value being sensed, the rate at which data is generated or polled may need to be adjusted. Over time, this rate could change dramatically based on the state of the phenomena being sensed. However, from a bandwidth management perspective, these are essentially constant bit rate (CBR) sources.

The other type of application is event-based, and does not send out sensor values (or some other value derived from the sensor reading) at regular intervals. Instead, the value and quantity of the traffic propagated through the network depends on whether or not the values being sensed, or how they change over time, represent an event. The notion of exactly what constitutes an "event" worth reporting is a function of

```
// timer fired to poll sensor
Timer.fired() {
  // obtain sensor reading
  val = Sensor.getValue();

  // transmit
  Network.send( val );

  // rate may change depending
  // on value
  new_rate = getNewRate( val );

  if (current_rate != new_rate) {
    current_rate = new_rate;
    Timer.setRate( new_rate );
  }
}
```

```
Sensor1.event(val) {
  processValue( SENSOR_1, val);
}

Sensor2.event(val) {
  processValue( SENSOR_2, val)
}

processValue( type, val )
{
  // record the reading and possibly
  // perform some analysis
  updateLocalState( type, val);
  event = getNextEventToSend( );
  if (event != DO_NOT_SEND)
    Network.send( event );
}
```

Figure 2: Pseudo-code representative of the class of applications with which our architecture is designed to interact. Left: a periodic sensing application. Right: an event-based sensing application.

the sensor type and the environment in which it is deployed. The right-hand column of Figure 2 shows an example of how an event-based sensor application might be structured. The essential point to note is that the sensing of the environment is entirely decoupled from the rate at which events are sent. Applications either poll their sensors or explicitly receive events, but this rate is generally much higher than the rate at which events are sent through the network. It is up to the application to decide how correlate multiple sensor readings and how the changes or levels of such sensor readings correspond to events.

Of course, these two types of applications will usually coexist, providing the basic building blocks for a wide range of monitoring systems. In addition, in response to data obtained from the sensor network, the monitoring station may respond by actuating events in the environment or by requesting changes in the nature and rate of the sensed data being reported.

When wireless network bandwidth is scarce relative to node computation, either because the transmission bandwidth is small or because it consumes a large amount of energy to transmit packets on the wireless link, bandwidth management is an important problem. We address this broad problem along three dimensions: dynamic bandwidth allocation based on the nature of sensed data and events, congestion control, and improving the capacity of the network using multiple SAPs and load-sensitive packet routing.

In this paper, we ignore energy-efficiency and focus on solely on bandwidth management. As such, some of our techniques are optimized by promiscuous packet reception and by nodes overhearing each other to implement suppression and obtain better information about the state of the network. These techniques interact poorly with some previously proposed energy-optimizing approaches. We don't explore these interactions further in this paper, leaving that for future work. However, in our indoor sensor deployment, we have found that it has been easy to connect all our sensors to nearby power outlets, and the primary practical bottleneck in our system is in fact wireless bandwidth. We expect this to be true of many other indoor sensor deployments as well.

### 2.2 Rule System

Rules allow applications and network administrators to specify how data packets should be treated by nodes in the sensor network. A rule maps packets, whose payload includes a sensor data type and a value being reported, to a desired reception rate and a traffic class. Specifically, a rule is a function $F$ that takes the sensor's data type, the value being reported, and other attributes (*e.g.*, sensor location, local node state corresponding to the stream, etc.) as input and produces a rate and traffic class.

$$F(\text{type, value, attr}) \longrightarrow \{\text{rate, class}\}. \quad (1)$$

The desired reception rate is an *aggregate* rate at which the monitoring application running "behind" the sensor access point wants data that matches the rule's precondition. The traffic class in a rule allows a network administrator as well as application writers to specify the relative importance of different data ranges for sensor streams. For example, one rule might specify that the temperature sensors provide an aggregate data rate of 4 packets/second, while a second more-specific rule might specify that temperature sensors that report values above 80 degrees from certain locations do so at

10 packets/second on a different traffic class that is treated at higher priority in the network. The semantics our system attempts to provide are that the aggregate rate at which data matching a rule reach the SAP nodes is no larger than the specified rate, with the messages delivered on the specified traffic class. In practice, these semantics are "best-effort" in that they may not always be met. We discuss this in Section 2.2.2.

A network administrator disseminates rules to nodes (*e.g.*, using a broadcast protocol, or by manually configuring the sensors before they are deployed), ordering the rules in precedence order. Whenever a node receives or generates a packet, it classifies the packet according to the rules and enqueues the packet on the queue corresponding to the traffic class.

#### 2.2.1 Rule syntax and semantics

Our rule system is simple—it applies simple logical expressions to packet fields and determines what to do with the packet. The formal specification is as follows:

| | |
|---|---|
| *rule* | ← [ *precond* ⟶ {rate class} ] |
| *precond* | ← *comparison* \| *precond* `OR` *precond* \| *precond* `AND` *precond* \| `NOT` *precond* \| (*precond*) |
| *comparison* | ← application *op* value |
| *application* | ← `light` \| `temp` \| `occupancy` . . . |
| *op* | ← `>` \| `<` \| `>=` \| `<=` \| `!=` \| `==` |

One can generalize the above rule format to include more complex rules that trigger actions, but we have found that even this restricted set is rather powerful and can lead to complex interactions between rules. In general, a given data packet can match multiple rules; our current approach to handling this is to take the action corresponding to the first rule that matches in the precedence order specified earlier. We may revisit this decision in the future when we gain more experience with the rule system.

Rules may interact with application computation running in the sensor network, especially with aggregate operators implemented at sensor nodes. When a node receives a packet, before consulting the rules to determine how to forward the packet, the node's network layer first determines if the application layer running at the node has registered interest in processing packets of that type, in which case the packet is delivered to the appropriate packet handler. The specification of application handlers is done using the same rule syntax described above, with the {rate, class} result being replaced with an application's input packet queue.

#### 2.2.2 Rule enforcement

Each node enforces rules in two ways: it implements a packet scheduling discipline to classify and forward packets in different traffic classes, and it implements a rate-control mechanism to throttle packets to the rates specified in their best-matching rule.

The semantics we associate with a traffic class are *delay priority* semantics rather than bandwidth sharing semantics. That is, we choose a simple static priority forwarding scheme where packets are sent from the highest-priority queue that is not empty. In an earlier prototype of our system, we implemented traffic classes using a weighted fair queuing discipline, which shares link bandwidth amongst the queues, but found that it was not well-suited to our network and applications. There are two reasons for this. First, the rules already specify packet rates, which are honored in best-effort fashion by the network, and a different scheduling mechanism to ensure bandwidth sharing does not interact well with the rate-control. Second, running WFQ does not allow important messages to be sent at higher priority than less-important ones, because it is hard to estimate *a priori* the bandwidth that might be required for messages at different levels of importance at any given time. When important events happen, our goal is to forward them on in preference to less-important data without "inversion," a task that is facilitated by running a simple priority scheduler at each link.

Each node controls the rate at which packets matching any given rule are sent on the network using two forms of *suppression*. For each rule, a node maintains the last time at which a packet was sent from the node to a SAP, as well as the size of the packet. When the node receives a packet, it forwards the packet on only if the time since the last transmission on the packet's matching rule was larger than the duration required by the specified rate; otherwise, the packet is simply dropped. The node thus implements, for every rule, a leaky bucket filter of depth 1.

Each node also implements *distributed suppression* by listening opportunistically to other transmissions and updating its local estimate of the last time a packet matching any of its local rules was sent. This is a useful optimization when there is spatial locality in rules, for instance when geographic location is used as a rule's attribute, or when there is a high degree of spatial correlation in a data stream. Both of these situations are common in many sensor networks.

If there were only one SAP in the network topology, the maximum rate at which any rule's packets are delivered won't exceed the specified rate, although with multiple SAPs this statement is not true. Of course, nodes that are on paths to different SAPs will still implement suppression independent of the SAP to which the packet is destined.

### 2.3 Hop-by-Hop Flow Control

Any network with competing and dynamically varying traffic requires some form of congestion control to avoid high packet loss rates, long queues, or congestion-triggered collapse. On the Internet, congestion control is done end-to-end at the transport in transport protocols like TCP or in end-system modules like the Congestion Manager [3].

End-to-end flow control schemes like TCP are not well-suited to our domain for the following reasons:

1. Mismatch with applications that send at constant periodicity, with occasional bursts. As explained in Section 2.1, many sensor network applications involve low-rate CBR flows that might experience a sudden increase in transmission rate when an interesting event occurs. With TCP, every incoming ACK causes an increase in the transmission window size. The problem is that if the stream wasn't actually saturating the network (as in a low-rate CBR), this window inflation is artificial and does not signify that the capacity indicated by the window is actually available! Now, when an event occurs that causes a sequence of packets to be sent in quick succession, TCP would assume that the large window was usable, but the result would be packet loss because the network isn't actually capable of sustaining this large window (rate).
2. End-to-end acknowledgment overhead. Many end-to-end congestion control schemes require ACKs to be sent from the receiver, to allow the sender to obtain an accurate idea of the state of the network. Many sensor streams don't require the reliability semantics of TCP, making the cumulative ACKs unnecessary. Furthermore, since most sensor data packets are small, end-to-end ACKs would consume a substantial fraction of the overall network bandwidth.
3. Bad performance when windows are small. Protocols like TCP are notorious for poor performance when windows are small [2]. Although some recent solutions have been proposed for this problem [1], a fundamental problem is that small-window paths often tend to cause high packet loss rates in the way TCP adapts while probing for more bandwidth.

Our solution to this problem is to revive an old idea—*hop-by-hop flow control*—and adapt it to sensor networks. This idea has been proposed before in a few different contexts, most notably high-speed networks where it was once believed that end-to-end approaches could not work well in large bandwidth-delay product networks.

The idea in hop-by-hop flow control it that a congested node provides essentially immediate feedback to an upstream neighbor sending packets to it about the onset of congestion. In our system, we take advantage of the ability to *synchronously* send small link-layer messages from a receiving node to a transmitting node. The conventional use of this feature is to send link-layer ACKs upon successful reception, but we use it to send *synchronous NACKs* (negative acknowledgments) when the instantaneous queue size at a node exceeds a high-water threshold. The idea is that the packet causing this NACK will be enqueued, but the notification will cause the transmitting node to slow down. With this slow-down, transmitting nodes upstream from this part of the network will soon throttle back as queues fill up. The key point is that this happens without packet loss.

When a node receives a synchronous NACK in response to a transmission, it throttles its rate of transmission to that neighbor. Our scheme determines how long to wait by overhearing the congested nodes transmissions, and resuming transmissions to the neighbor after it hears at least two packet transmissions from the neighbor. This indicates that two packets of space have cleared the neighbor's queue.

*Implicit ACKs* are an alternative to using synchronous NACKs. A node receives an implicit ACK when it hears its downstream neighbor forward a recently-sent packet. We considered this alternative, previously explored by Woo and Culler [19] in a different rate control proposal, but found the following problems with it. First, if nodes have queues of even a few packets in length, a packet enqueued near the tail of the downstream queue may take a variable amount of time depending on the available bandwidth to actually be transmitted. This implies both that a non-trivial timeout needs to be set, and that the feedback could take as long as the queue depth to arrive at the upstream node. The second problem is even worse—this method does not work when the downstream node's application layer suppresses packet (*e.g.*, because it is doing some aggregation). The synchronous-NACK solution does not have these problems.

By using the hop-by-hop scheme with synchronous NACKs, the expectation is that once a packet is admitted into the system, it is generally not dropped due to congestion.[1] Drops only occur due to data transmission errors or through explicit suppression by application-layer computation at nodes. This is a useful feature in wireless networks where network bandwidth is a scarce resource, and where it is wasteful to send packets only to have them dropped at later points on their path. Our performance results in Section 4 show that this approach uses available bandwidth prudently.

### 2.4 SAP Selection

A large sensor network connected using a single SAP rapidly becomes unusable because the link connecting the SAP to the rest of the network becomes the bottleneck. A solution to this capacity problem is to use multiple SAPs. This leads to the SAP selection problem: how does a node decide which SAP

[1] Even if a drop is required because a downstream queue is full, a downstream node preemptively drops a lower-priority packet from its input queue when a higher-priority packet arrives at a full queue.

to send its data to at any point in time?

Every node in the sensor network needs to select a "good" path to send data toward an access point. This is a standard routing problem, but we are interested in not the best path to a given destination node but the best path to any one of several possible APs. Previously, researchers have shown that a pure hop-count metric for path selection, standard in wired routing protocols, is not appropriate for wireless networks because picking a path with the smallest number of hops tends to pick marginal links that are maximally separated from each other, whose link quality is therefore bad [5, 21].

One potential solution to this problem, explored by Yarvis *et al.*, is to pick the path with the largest end-to-end probability of success. While this helps individual nodes pick the best paths, it does not take into account the *cost* to the network of nodes picking these paths. Because this metric prefers a long path with low packet loss probability over a shorter path with higher loss probability to an AP, it introduces a larger load into the network by requiring a larger number of wireless transmissions. This reduces the overall capacity of the wireless network, because the capacity decreases as the average number of hops traversed by the communicating nodes grows [12].

In our network, nodes don't retransmit lost packets at the link-layer, as in 802.11. Furthermore, because our network uses hop-by-hop flow control to reduce congestion-related packet losses, proactively avoiding paths that have a higher traffic load is worthwhile. On any path, a measure of the *benefit* of using a path is proportional to the end-to-end packet success rate, and inversely proportional to the aggregate traffic load along the path. The *cost* of using the path, in the form of increased bandwidth used, is proportional to the number of hops it has. Together, the ratio of cost to benefit gives us the following path metric, $M_P$ for a path $P$:

$$M_P = \frac{\#\text{hops}(P) \cdot \max_P \text{Load}}{\text{path\_success\_prob}} \quad (2)$$

#### 2.4.1 Routing protocol

Each node picks the path $P$ with smallest value of $M_P$. Disseminating the information required to pick an appropriate path is easy to do using a distance-vector protocol, because the three components of the metric—"hops", "load", and "path success probability"—are all associative metrics. Each node propagates these three different quantities in distance-vector fashion corresponding to the path to the node's current estimate of the "best" SAP.

Each node receives routing announcements from its neighbors and maintains the identity of its current "parent" that it will use to forward packets to its "best" SAP according to the metric $M_P$. The node *does not* explicitly maintain the identity of any SAP, because it is interested in "anycasting" its data to the best SAP. Because of this, our approach scales well as the number of SAPs increases, without requiring any additional routing state at any node. We show how this is done in Section 3.3. To avoid routing loops, the protocol uses the increasing sequence number solution proposed in DSDV [17].

#### 2.4.2 Avoiding load-induced oscillations

Load-based routing systems can suffer from load-induced oscillations, when traffic shifts between two or more paths because the system always finds an unused path to be better, only for it to become worse when traffic shifts to use that path. This is not a new problem, and is one reason why such approaches have proved difficult to implement on the Internet.

We use a simple heuristic to alleviate the severity of the problem in our context. For each node $N$, the load that it advertises to its neighbors is the maximum number of packets per second traversing any link from $N$ to the SAP it is associated with. The load that it uses to calculate $M_P$ is that figure, *minus* the number of packets per second that it sources onto the network. This prevents the load term as used in the metric calculation from changing just because a node sources traffic onto the network. The key idea is for each node to *not* use the contribution to a path's load caused by the node itself.

## 3 Implementation

We selected the Berkeley *Mica* Mote sensor network node on which to implement these bandwidth management techniques. Our implemented system is called *Mist*. The Mica Motes are equipped with a 4 MHz Atmel ATMEGA 128 micro-controller, a 40 Kbps ASK radio, and can be attached to a variety of sensor boards. We run the TinyOS [9] operating system on the Motes, a tiny event-driven operating system written for the Motes. TinyOS and the Mist codebase are both written in NesC [6], a language resembling embedded C.

### 3.1 Rules

To implement the rule manager, we constructed a parser that accepts our rule grammar as specified in Section 2.2. The output of the parser is an array of *opcodes*, one corresponding to each step in the evaluation of the rule antecedent. These opcodes may be transmitted from a SAP to nodes in the sensor network, or may be statically-linked with the Mist codebase running on the Mote. For simplicity in implementation, we chose the latter, but for ease in changing the rule opcodes, we plan to implement the former in the future.

### 3.2 Flow Control

To implement our hop-by-hop flow control protocol, each node monitors the number of packets $B$ backlogged in a transmission queue of size $Q$. When $B < Q/2$, it replies to route-through traffic sent to it from other nodes with synchronous acknowledgments (ACKs) at the MAC layer. When $B \geq Q/2$, it replies with synchronous negative acknowledgments (NACKs).

The reception of an ACK has no impact on flow control. Upon receiving a NACK from a downstream parent, a sending node shuts off its transmission to that parent for a period of time. The node attempts to resume transmissions to the parent when it hears two packets sent by the parent, or after one second, whichever is smaller.

Since NACKs and ACKs are synchronous, they are only broadcast at the tail end of a packet transmission. This negates the need for a preamble or other such overhead to synchronize the senders and receivers. The side effect of this particular implementation is that other nodes sending to the same parent cannot take advantage of this feedback mechanism. In order implement the flow control mechanism exactly as described in the previous section, NACKs should be broadcast to all children. This feature was left out for simplicity, but our approaches matches the desired results very closely.

### 3.3 SAP Selection

SAPs are selected based on the metric described in the Section 2.4. To implement this, each node maintains a table of potential parents that can be selected to forward traffic. Periodically, for each neighbor, a node calculates this metric and selects the neighbor with the minimum value as the parent. Nodes construct this table by promiscuously listening on the channel for activity from neighboring nodes.

To facilitate the calculation of the routing metric for each neighbor, every outgoing packet of a node is tagged with information in addition to that which is already present in the TinyOS header. These fields include: source address, sequence number, hop-count, path-load, path link quality, priority, SAP ID, origin node address, and parent address. These values are recorded for each neighbor in our table (using the source address field as the key), to be used by the node when recalculating the metric.

In order to calculate the load parameter of the metric for a given neighbor, a node must maintain an estimate of the load experienced by the SAP with which it is currently associated. SAPs advertise a moving average (EWMA) of the number of packets forwarded over a 10-second window in the "path-load" field of the packets they transmit. All nodes that are associated with a given SAP propagate this value in the path-load field of the packets they transmit. When it comes time for a node to calculate the load parameter of the metric for each of the its neighbors, the node relies on the advertised path load value from each of its neighbors to make this calculation. If the neighbor for which we are currently calculating the metric is associated with a different SAP (as dictated by the last value of the SAP id field) than our parent, then the load parameter of the metric is simply the advertised path-load. However, if the neighbor is associated with a different SAP, the local node's contribution to this path-load level must be subtracted, as explained in Section 2.4.2. The local node's contribution to the path load is simply an EWMA of the number of packets forwarded over a 10-second window.

Each node calculates the path success rate parameter of the metric for a given neighbor by multiplying the path success rate advertised by the neighbor in its path link quality field with the node's own estimate of the single-hop link quality to that neighbor. Each packet sent on any link contains a link-specific monotonically increasing sequence number (which wraps-around). A neighbors can then infer one-hop link quality based on gaps in the received sequence number space from packets it overhears on *any* link. This value represents the link quality along the direction opposite to the usual flow of data, because data would flow from the overhearing node toward the node it heard from. Despite this asymmetry, the information is useful at obtaining reasonable estimates. Our solution does not currently handle asymmetric losses adequately.

Finally, to calculate the hop-count parameter of the metric for a given neighbor, a node uses the value last advertised in the hop-count field for a packet sourced by that neighbor. When nodes transmit a packet, the hop-count field is set to be one greater than that of the node's parent.

Since the Motes lack a floating point unit, our implementation only uses integer arithmetic. It scales link quality to a value between 0 and 128. Rather than directly calculating the metric, the implementation calculates the logarithm of the metric. This allows us to use additions in place of multiplications and subtractions in place of divisions while comparing metrics to make a choice of smallest one to pick.

We have not yet implemented the loop-avoidance algorithm using increasing sequence numbers *a la* DSDV. Instead, for every packet originated by the node, the origin and parent address fields of the packet header are filled in to their corresponding values.[2] This allows parents to discover their directly connected children; parents will never include a child among the set of nodes that could become parents. Such a scheme avoid 2-hop loops, but larger loops could still exist. However, this has not yet proved to be much of a problem in practice, but it is clear that implementing the loop-avoidance method will make the system more robust.

Our implementation is somewhat heavy-handed in its distribution of link metric information. Fields such as hop-count

[2]These fields are also used to provide a visualization of the network to facilitate better analysis.

should not change very frequently. As a result, many of the fields included in the header could instead be transmitted using explicit routing messages, reducing the header overhead. However, for the purposes of our experiments, the benefits of having constantly updated routing information outweighed the loss of payload bandwidth.

# 4 Experiments

This section presents experimental results to validate each of our three design components: the rule system, static priority-based queuing, and our SAP selection metric.

## 4.1 The Rule System

We now experimentally show how the rules described in Section 2.2 enable traffic to be mapped onto different bandwidth classes, which in turn enables important events to be given priority both at the network layer in queues, and at the MAC layer.

### 4.1.1 Static Priority-Based Queuing

In these experiments flows compete with another for the common bandwidth to a SAP. We run these experiments on a small test topology, unrelated to our larger 40-node testbed discussed below. Our rule-based system sets the priority for each Motes' traffic, based on information that the Mote senses. We first turn off Mote rules, and stimulate four Motes to send to a common SAP. The Motes send through a common bottleneck node. The results are shown in Figure 3. As we see in the figure, the Motes share the bandwidth equally between them.

Next, we enabled Motes' rules. We simulated an important event near nodes 61, 63, and 68. Node 60 continued to send unimportant traffic. The results are shown in Figure 4. The low-priority flow gets much less of the bandwidth because the high-priority flows' packets evict the low-priority flows' packets in the bottleneck node's transmission queue.

The two foregoing experiments show how a static priority queuing discipline can give important traffic preference over unimportant traffic. We now turn to MAC-layer support, to add further support for preferential service of important traffic.

### 4.1.2 MAC Layer Support

Although allowing high-priority packets to overwrite low-priority packets when buffer space runs out can lead to an improvement in high-priority streams' throughput, MAC-layer support is needed to ensure that if any high-priority packets are waiting to be sent, their nodes will win any contention

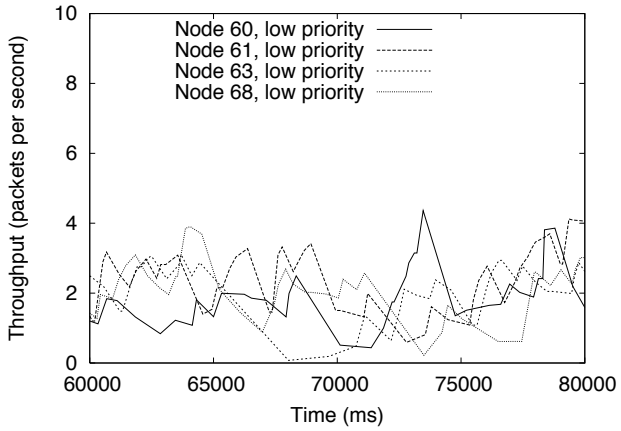


Figure 3: Instantaneous throughput achieved by Motes operating with rules disabled. There are four senders in this experiment, all transmitting as fast as possible to a common SAP, through a fifth common bottleneck node.

| Metric | Receive rate | Throughput (pps) |
|---|---|---|
| SAP Selection | 0.919 | 14.67 |
| Minimum hop-count | 0.749 | 11.23 |

Table 1: A three-node experiment showing the benefit of using the SAP Selection metric on receive rate and throughput.

competition at the MAC layer. In this section, we show how the MAC layer can be modified to accomplish exactly that.

We configured three Motes to send packets as fast as possible to a common SAP. The traffic that the Motes sent was classified into high and low priorities based on a field in each packet's header. Nodes 1 and 2 were configured to sent 90% high-priority traffic and 10% low-priority. Node 3 was configured to send only low-priority traffic. The black bars in Figure 5 show the desired allocation of bandwidth: the high-priority traffic from nodes 1 and 2 should consume all of the channel, and be split evenly between the two. The dark grey bars in the figure show the performance of our implementation, which comes very close to the desired bandwidth allocation. The light grey bars show that without our modifications, low-priority traffic would consume some of the high-priority traffic's bandwidth.

## 4.2 SAP Selection

Our first experiment demonstrates the drawbacks of using a simple hop-count metric for SAP selection. We placed three Motes in a linear formation in an office environment. We designated one of the three motes on the end of the linear chain as the SAP. Both other motes attempted to send data to the SAP. There was a marginal link from the far Mote to the SAP. Using our SAP selection metric, we were able to avoid the

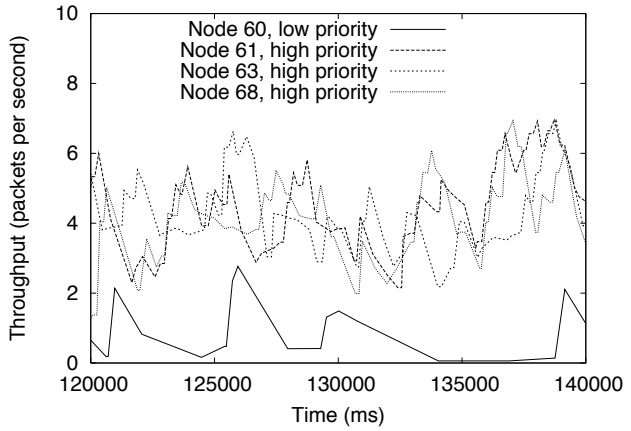


Figure 4: Instantaneous throughput achieved by Motes operating with rules enabled. The setup for this experiment is identical to the one in Figure 4.

marginal link, and forward packets from the far Mote over the two-hop link. Using the minimum hop-count metric, both Motes sent over their respective one-hop routes. Even though the SAP selection metric was forwarding packets over two hops instead of one, the marginal link from the SAP to the far Mote was bad enough that the SAP selection metric resulted in better throughput and receive rate. This unsurprising result forms the basis for an intuition for why our SAP selection metric performs better in the next experiment, a test situated in an 40-node testbed.

#### 4.2.1 Testbed Experiment

In order to exhaustively test the SAP selection metric, we conducted an experiments using a large sensor network testbed. The layout of the sensor network is described in Figure 6. We placed 40 nodes throughout one floor of an office building. Total aggregate throughput for the SAPs was recorded for 200 seconds. We tested:

- The SAP selection metric, as described in Section 2.4.
- The hop count metric, which minimizes the number of transit nodes traversed until a SAP is reached.

In addition to varying the path selection metric, the number of SAPs available to the nodes was varied between one, two, and three. Each node in the network sends a 36-byte test probe message every two seconds. Figures 7, 8, and 9 shows the number of such messages that each SAP received when there are one, two, and three SAPs, respectively. These figures show that our metric allows more nodes to successfully reach the SAP.

The results of the experiment are summarized in Figure 10. Independent of the number of SAPs, our metric outperforms

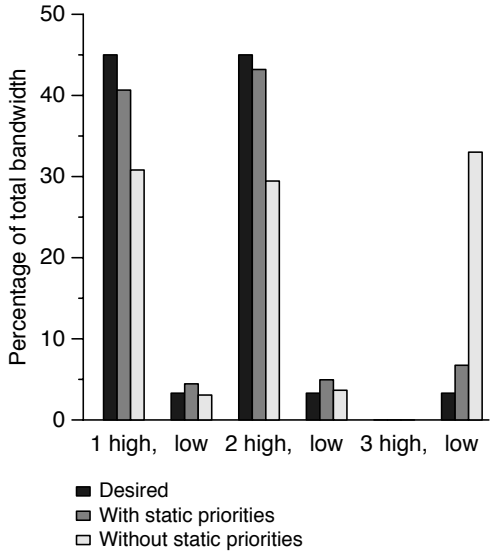


Figure 5: static priorities are good

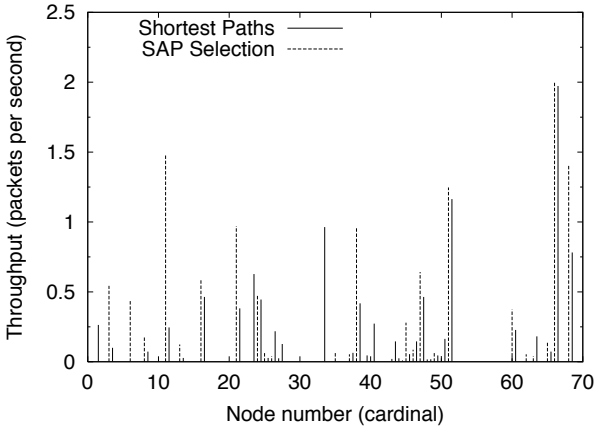


Figure 7: Throughput in a 40-node, single-SAP testbed, broken down by node number. For each node, we plot the delivered throughput in packets per second in our testbed. The offered load is two 36-byte packets per second. The x-axis of the "Shortest paths" results is shifted by one half-unit for clarity.

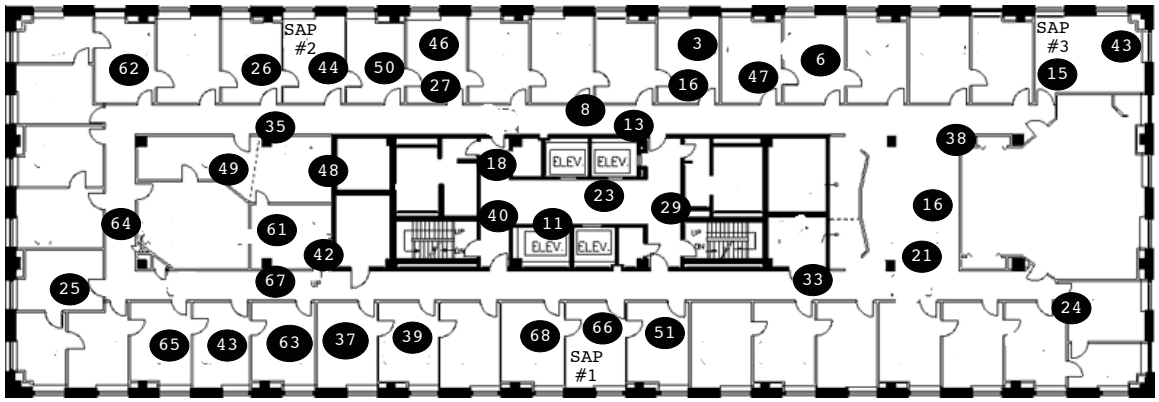


Figure 6: Our testbed topology. This map shows where we placed each Mote on an office floorplan.

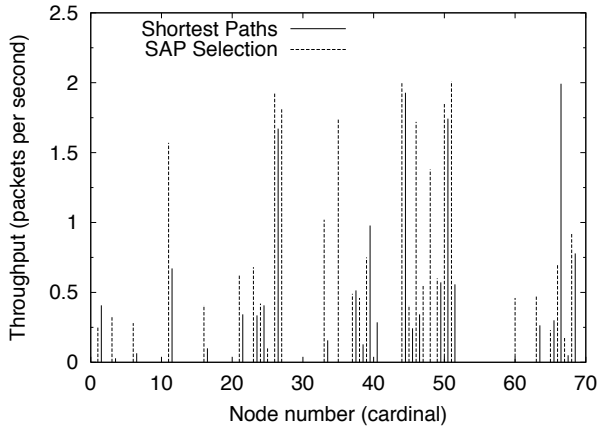


Figure 8: Throughput in a 40-node, two-SAP testbed, broken down by node number. We run the same experiment as in Figure 7, but use two SAPs instead of one.

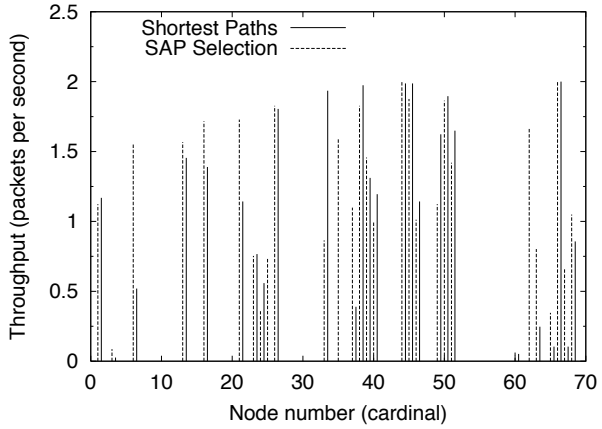


Figure 9: Throughput in a 40-node, three-SAP testbed, broken down by node number. We run the same experiment as in Figure 7, but use three SAPs instead of one.

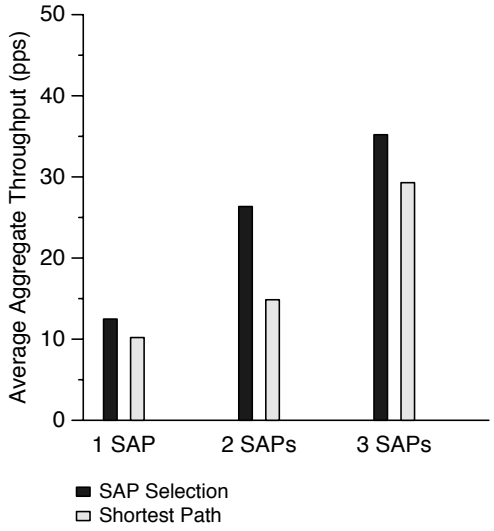


Figure 10: Effects of the SAP selection metric on the aggregate throughput of the network. SAP selection improves throughput to varying degrees depending on the number of SAPs deployed.

simple hop count. In addition, it is clear that adding SAPs to sensor networks improves the network capacity.

## 5 Related work

### 5.1 Congestion control and routing

As part the design of a MAC protocol for sensor networks, Woo and Culler propose a rate control method [19]. Their proposal is to use adaptive rate control at each node using additive-increase and multiplicative-decrease (AIMD). When a node finds that a packet it previously sent was forwarded on, then it increases the transmission rate additively to the neighbor. When it does not hear a previous transmission being successfully forwarded (presumably after a timeout), it

reduces the transmission rate to that neighbor. This scheme differs in several important details from ours. First, we use synchronous NACKs if queue space is unavailable at a downstream node, which provides much faster feedback about congestion. Second, because nodes overhear other transmissions, they maintain good information about the congestion state at their parents. Our scheme does not require any approximate estimation that schemes like AIMD require.

Hop-by-hop flow control protocols have been extensively studied in the context of ATM and Gigabit Ethernet [16, 11, 14, 15]. The motivation in these high-speed networks is to avoid the burst behavior of end-to-end protocols like TCP at small round-trip times. In sensor networks, hop-by-hop flow control is attractive because it allows good congestion adaptation without incurring losses or requiring the expensive end-to-end acknowledgments that are unnecessary for many streams that don't require TCP-style reliability.

Our rule-based approach to sensor network bandwidth allocation does not seem to have directly related previous work. Madden *et al.*'s TAG sensor query processing system implements aggregates inside the sensor network [13], and can be used in conjunction with our rules. In traditional Internet routers, classifiers and policy rules are used to effect both forwarding and routing decisions.

Sensor information dissemination protocols like Directed Diffusion [10] and Spin [8] route packets according to node interest but do not take congestion, load, or packet loss into account. As described in Section 2.4, Yarvis' *et al.*'s tree-based routing metric considers loss rate and optimizes path success probability [21].

### 5.2 Energy-efficient protocols

This paper addresses problems caused by scarce network bandwidth and ignores energy consumption. A complete system will of course need to optimize both energy and bandwidth, and it would be interesting to study which of our techniques are problematic for energy consumption. For example, some of our techniques are optimized by nodes overhearing each other's transmissions, which requires nodes to be on and listening on the wireless channel when they might instead sleep to save energy. It might be possible to run topology formation algorithms like Span [4], GAF [20], or LEACH [7] that produce and change topologies on slower time-scales, and our bandwidth management schemes on each individual instance of the network topology.

Woo and Culler [19] compared the performance of various contention window-based MAC schemes, varying carrier sense time, backoff increase/decrease policies, and transmission deferral policies. All of their protocols used contention windows with the uniform distribution. Their evaluation of these conventional protocols was exhaustive, and leads us to our present design point. They also noted that 802.11 is energy-inefficient, since it listens throughout its backoff period, and its backoff period can grow to be quite long.

Ye *et al.*'s S-MAC [22] is a MAC protocol designed for saving energy in sensor networks. It uses periodic listen and sleep, the collision avoidance facilities of 802.11, and overhearing avoidance to reduce energy consumption. Singh *et al.*'s PAMAS [18] reduces energy consumption by powering off nodes when they are about to overhear a transmission from one of their neighbors. These schemes may be incompatible with some of our techniques that rely on promiscuous packet reception to improve bandwidth usage. A detailed study of the trade-offs between bandwidth-efficiency and energy-efficiency in shared-medium wireless sensor networks is an important topic for future work.

## 6 Conclusion

This paper presented three techniques to manage wireless network bandwidth in sensor networks. These techniques are tuned for a broad class on monitoring and control applications, where software running on general-purpose computers "pull" information from remote sensors and "push" actuations into the network. We described a simple rule-based approach to dynamically allocate bandwidth to sensor streams, a hop-by-hop flow control protocol to control congestion, and a load- and loss-sensitive SAP selection protocol that improves the overall capacity of the sensor network.

We have implemented these techniques in the TinyOS environment, and have conducted several experiments on a 40-node indoor wireless sensor testbed. Our results show that we can achieve an experimental increase in network capacity of up to 100%, while simultaneously reducing loss rates from 25% to 9% on some links.

A few important principles come out of our work. First, when wireless bandwidth is scarce, as is common in many sensor networks, it is important to maximize the "value" of each packet accepted into the system for transmission. This means that it is better to not accept the packet at the source, rather than accept it only to drop it later along the path. We find that hop-by-hop flow control is a simple and effective way to achieve this goal. Second, when events of interest occur, bandwidth needs to be shifted dynamically. We believe that our rule-based system provides an effective method to accomplish this in many cases. Third, achieving good capacity requires multiple SAPs and corresponding routing and SAP selection methods. While we do not claim that our proposal is optimal (or indeed even understand all the interactions well enough), we believe that our path selection metric combining hop-count, path load, and path success probability, is a good starting point and has advantages over traditional hop-count-based routing metrics.

This paper did not address energy-efficiency, focusing instead

solely on bandwidth management. We made this choice in order to understand the bandwidth issue in isolation, and also because our indoor sensor network runs using power sockets on walls. An important direction for future work is in understanding how techniques for optimizing these two different resources interact with each other.

## References


[1] ALLMAN, M., BALAKRISHNAN, H., AND FLOYD, S. *Enhancing TCP's Loss Recovery Using Limited Transmit*. Internet Engineering Task Force, Jan. 2001. RFC 3042.

[2] BALAKRISHNAN, H. *Challenges to Reliable Data Transport over Heterogeneous Wireless Networks*. PhD thesis, Univ. of California, Berkeley, August 1998.

[3] BALAKRISHNAN, H., RAHUL, H., AND SESHAN, S. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)* (Cambridge, MA, 1999), pp. 175–187.

[4] CHEN, B., JAMIESON, K., BALAKRISHNAN, H., AND MORRIS, R. Span: An Energy-Efficient Coordination Algorithm for Topology Maintainence in Ad Hoc Wireless Networks. In *Proceedings of the Seventh International ACM Conference on Mobile Computing and Networking (MOBICOM)* (Rome, Italy, July 2001), pp. 85–96.

[5] DE COUTO, D. S. J., AGUAYO, D., CHAMBERS, B. A., AND MORRIS, R. Performance of multihop wireless networks: Shortest path is not enough. In *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)* (Princeton, New Jersey, October 2002), ACM SIGCOMM.

[6] GAY, D., LEVIS, P., VON BEHREN, R., WELSH, M., BREWER, E., AND CULLER, D. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (San Diego, CA, June 2003).

[7] HEINZELMAN, W., CHANDRAKASAN, A., AND BALAKRISHNAN, H. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)* (January 2000).

[8] HEINZELMAN, W., KULIK, J., AND BALAKRISHNAN, H. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of the Fifth International ACM Conference on Mobile Computing and Networking (MOBICOM)* (Seattle, WA, 1999).

[9] HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. System Architecture Directions for Network Sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems* (Cambridge, MA, 2000), pp. 93–104.

[10] INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth International ACM Conference on Mobile Computing and Networking (MOBICOM)* (Boston, MA, 2000), pp. 56–67.

[11] LEE, D., COLERI, S., DONG, X., AND ERGEN, M. FLORAX—Flow-Rate Based Hop by Hop Back-pressure Control for IEEE 802.3x. In *5th IEEE International Conference on High Speed Networks and Multimedia Communications* (Jeju Island, Korea, July 2002).

[12] LI, J., BLAKE, C., DECOUTO, D. S. J., LEE, H. I., AND MORRIS, R. Capacity of Ad Hoc Wireless Networks. In *Proceedings of the Seventh International ACM Conference on Mobile Computing and Networking (MOBICOM)* (Rome, Italy, July 2001), pp. 61–69.

[13] MADDEN, S., FRANKLIN, M., HELLERSTEIN, J., AND HONG, W. TAG: a Tiny AGregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Boston, MA, December 2002).

[14] MISHRA, P., AND KANAKIA, H. A Hop by Hop Rate-based Congestion Control Scheme. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)* (Baltimore, MD, August 1992), pp. 112–123.

[15] NOUREDDINE, W., AND TOBAGI, F. Selective Backpressure in Switched Ethernet LANs. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)* (Rio De Janeiro, Brazil, December 1999), pp. 1256–1263.

[16] ÖZVEREN, C., SIMCOE, R., AND VARGHESE, G. Reliable and Efficient Hop-by-Hop Flow Control. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)* (London, UK, August 1994).

[17] PERKINS, C., AND BHAGWAT, P. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *Computer Communication Review* (October 1994).

[18] SINGH, S., WOO, M., AND RAGHAVENDRA, C. S. Power-Aware Routing in Mobile Ad Hoc Networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)* (Dallas, TX, 1998), pp. 181–190.

[19] WOO, A., AND CULLER, D. A Transmission Control Scheme for Media Access in Sensor Networks. In *Proceedings of the Seventh International ACM Conference on Mobile Computing and Networking (MOBICOM)* (Rome, Italy, 2001).

[20] XU, Y., HEIDEMANN, J., AND ESTRIN, D. Geography-Informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the Seventh International ACM Conference on Mobile Computing and Networking (MOBICOM)* (Rome, Italy), pp. 70–84.

[21] YARVIS, M., CONNER, W., KRISHNAMURTHY, L., MAINWARING, A., CHABRA, J., AND ELLIOTT, B. Real-World Experiences with an Interactive Ad Hoc Sensor Network. In *Proceedings of the IEEE International Conference on Parallel Processing Workshops* (Vancouver, BC, August 2002), pp. 143–152.

[22] YE, W., HEIDEMANN, J., AND ESTRIN, D. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)* (New York, NY, June 2002), pp. 1567–1576.