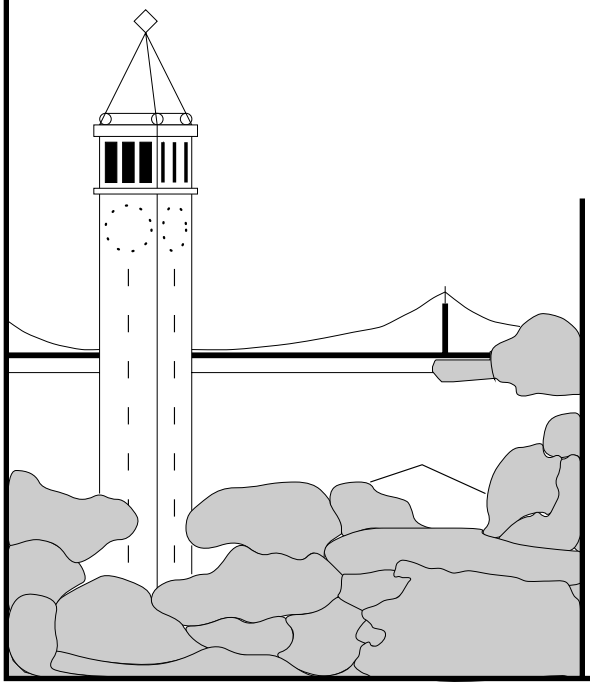
Self-Tuning Energy-Aware Multichannel (STEAM) Scheduling

Umesh Shankar



Report No. UCB/CSD-4-1300

March 2004

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Self-Tuning Energy-Aware Multichannel (STEAM) Scheduling

Umesh Shankar
University of California, Berkeley
ushankar@cs.berkeley.edu

1 Introduction

1.1 Sensor Networks

The emerging field of sensor networks represents a new domain of applications and networks. The standard model consists of a set of very small *sensor nodes*, each with very limited resources (KBs of memory, a few MIPS, 8-bit CPU). These nodes may be scattered about; each takes periodic readings of its environment using some particular sensor–e.g., temperature or humidity–and relays that information via a multihop ad-hoc network to a *base station*. The base station is assumed to be a powerful (say laptop-class), trusted node that does aggregation and other processing.

Power management is of considerable importance in the sensor regime: networks are expected to last for months or years without battery replacement or charging. Since the sensors' radios are often the dominant consumer of energy, it is vital that we limit their use: energy may be saved by turning the radio off when it is not needed. Thus two key challenges are minimizing *idle listening*, listening when no message is being sent, and minimizing *useless overhearing*, listening to messages destined for another node. Ideally achieving these goals should not come at the expense of large increases in sending time or collision probability.

1.2 Our Contribution

The main contribution of this paper is a new scheme for communication scheduling. It can schedule multi-channel communication, i.e., multiple frequencies or DSSS codes combined with a TDMA-like scheme for coordinating rendezvous times. The scheduling is done dynamically and in a local, distributed fashion and responds to changing radio conditions, node joins and leaves, and traffic rates. The two main challenges are doing channel allocation and efficient state exchange. The latter problem is made more difficult since ordinary overhearing is not possible in a multichannel system. The algorithms presented require only a small amount of soft state to be kept. We present analytic results and simulation results for new algorithms for each of these. We also present straightforward extensions to perform traffic-based adaptation and to achieve per-node fairness.

2 Goals and Assumptions

2.1 Goals

Our routing mechanism was designed with the following ultimate goals in mind:

- Reduce power consumption by turning radios off.
- Gracefully handle node joins, leaves, and dynamically changing connectivity.

- Have some provisions for fairness, assuming that nodes comply with the protocol.

In particular, we did not try to secure our protocol against malicious attackers. Our goal was more to explore the limits of power saving; we believe future versions of the protocol could then make tradeoffs as necessary to achieve more security at the cost of energy use.

We also wanted the protocol itself to have certain properties to facilitate practical implementation:

- The routing protocol should have a very small extra message overhead. After all, minimizing radio use is our primary goal.
- Since nodes are severely resource constrained, each node should have to maintain very little state and perform a very small amount of computation to make scheduling decisions.
- The algorithm should be as localized as possible for quick response and lower overhead traffic.

2.2 Assumptions

In order to do any kind of analysis, we had to make precise the capabilities of the nodes, the network, and services we assume are available. We also assume certain characteristics about the application being run in terms of the traffic patterns it generates. The properties below are typical of a large class of applications, including building and habitat monitoring.

- **Carrier Sense ability.** We assume that the radio MAC layer can do carrier sense. This scheme is easier to implement and has less overhead than RTS/CTS. Although it suffers from the hidden terminal problem, hidden terminals are accounting for in each node's parent link estimation.
- **Local time synchronization.** Each node must be synchronized with its neighbors on the scale of a few milliseconds. Work by Elson and Estrin[EE01] suggests that this is feasible; they achieved global synchronization on the order of microseconds. Rather than running a separate protocol, we maintain synchronization by piggybacking the relevant information onto data messages and link-layer acknowledgements.
- **Tree-based routing.** We assume that messages generated in the network are sensor readings that are sent towards the base station (or the root of some other tree). One notable exception is acknowledgements; we have provisions for link-layer (not end-to-end) ACKs. Point-to-point communication is not explicitly considered.

3 Algorithm Detail

3.1 Overview

The ideal communication schedule (assuming a network with sufficient bandwidth, which is not always the case) would be one such that each node only listened when there were messages being sent to it, each node only sent a message when the receiver was listening, and there was no overhead (control data) being exchanged. Naturally, it is not possible to achieve this in practice since many variables are unpredictable: a change in the environment may trigger a higher volume of messages or create a new radio interference graph, for example. Nonetheless, our algorithms attempt to approach this ideal, modifying the schedule only when some condition is not being satisfied and it is thought that a change would improve the situation.

In our scheme, each node determines a rendezvous time at which nodes may send messages to it. These decisions are made unilaterally, so there is no agreement necessary. There is, however, a shared control

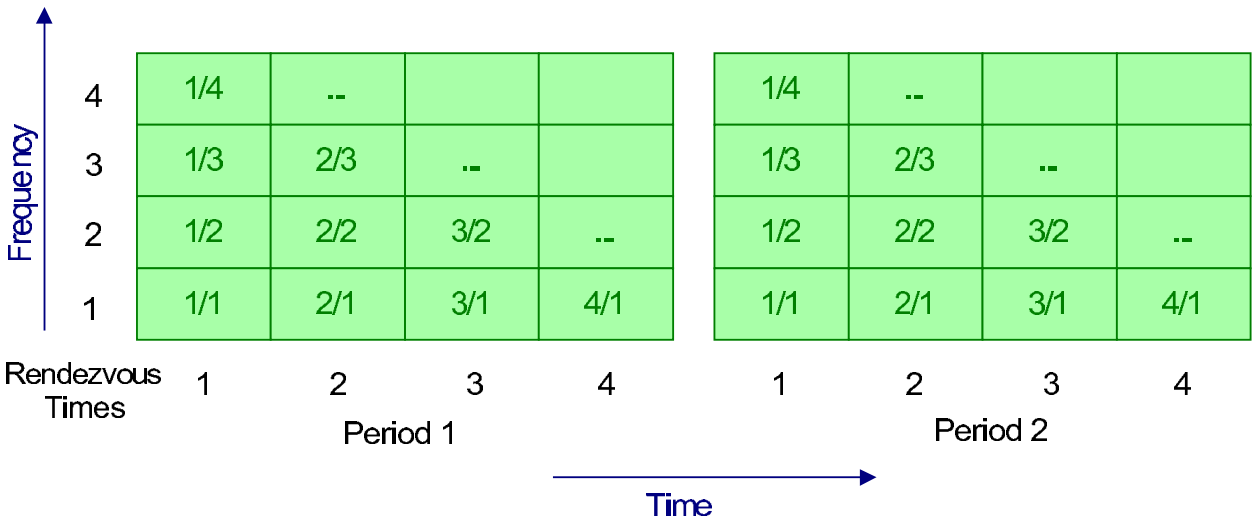


Figure 1: **The relationship between channels (inside the boxes), periods, rendezvous times (repeating in time on the horizontal axis), and frequencies or DSSS codes (vertical axis).**

channel through which information about the choices is distributed. Thus we use the broadcast nature of the channel to our advantage when it makes sense, viz., for state exchange, while trying to minimize the amount of unnecessary listening time during all other periods. Energy is minimized by determining a schedule that minimizes conflicts and by minimizing state exchanges needed to achieve such a schedule.

In the following sections, we will detail a basic version of the algorithm, later adding provisions for fairness and adaptive listen times, trading longer latency for lower energy consumption.

3.2 Multiple Channels

First, we define a *channel* to be some pair of (frequency or DSSS code, rendezvous time) (see Figure 3.2). The discrete rendezvous time offsets repeat periodically. Each node chooses a *receive-only* channel for itself. The node then agrees to listen using the specified frequency at the specified time during every period. If it hears a message starting, it remains on until it has received the entire message; otherwise it turns off. Thus each node only turns on long enough to determine that no start symbol is being sent when there are no messages to be sent to it. In particular, no reservations are required between senders and receivers; the senders may send on the receiver's channel during any period.

One design question is how long each period should be: the minimum spacing of rendezvous times is some constant dictated by the node's clock drift and the radio's ability to do carrier sensing, but there is no fundamental limit on the number of rendezvous times per period. The tradeoff here is between having more channels (and thus a lower chance of conflict) with longer periods versus having a smaller transmission delay with shorter periods. In general, then, we should choose the longest period length possible within our application's delay tolerance.

3.3 Channel Allocation and Dynamic Switching

3.3.1 Partial conflict

The question then is: how do we allocate channels to nodes so as to avoid useless overhearing? The problem of channel allocation is akin to that of graph coloring, with the channels as colors. As we shall see, however,

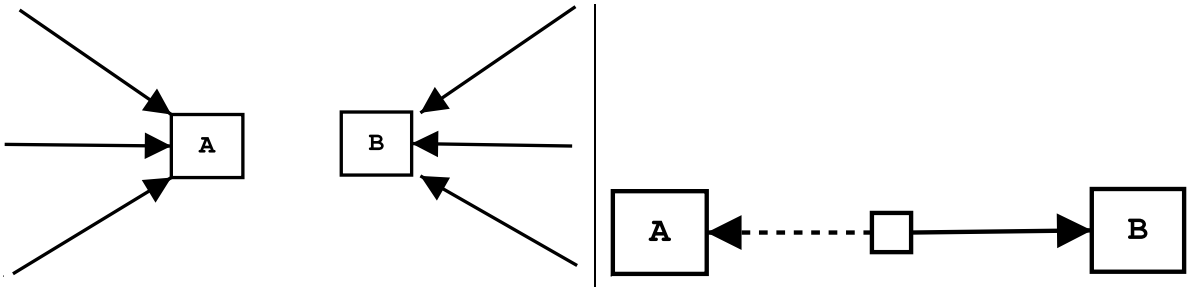


Figure 2: (a) Two nodes (A and B) may be in range of each other and still receive on the same channel, if their children are far apart. (b) Two nodes (A and B) may be out of range of each other and still have a channel conflict. In this case B's child is in range of A. This is also known as the hidden terminal problem.

it is not really the graph coloring problem. We will require some new machinery to solve the problem.

The first problem with the graph coloring analogy is that the notion of conflict, i.e., two adjacent nodes having the same color, is neither symmetric nor well-defined here. For the following discussion, the receive channels being chosen by the nodes will correspond to the colors we are using to color the graph.

In our first try, let us say that edges in the graph are defined by radio connectivity. The first thing to notice is that two adjacent nodes may select the same channel, but if each's children (nodes sending messages to them) are not in range of the other, there is in fact no conflict (see Figure 2a). Likewise, two nodes not in range of each other may conflict if their children are between them (see Figure 2b). So: two nodes conflict if one has children that are capable of causing radio interference for the other.

Let us therefore next consider a graph such that there is an edge between nodes A and B if a child of either one is within range of the other. We might try to color this graph, and if we can, then we are assured of a conflict-free schedule. Such a graph, though, contains many edges since the rule for adding edges is weak; it may not be colorable. On the other hand, if two nodes each have five child nodes and, say, only one of A's children is the cause of the conflict, then the situation may be tolerable. Node A will not listen to any messages destined for it, and Node B will only hear one such out of six. Thus we arrive at the notion of *partial conflict* which captures the notion that two nodes may sometimes conflict, but not always. In this case, it depends on how many children of one node are in range of another with the same receive channel. Partial conflict is one reason that straightforward graph coloring is insufficient. Later we will develop a cost-benefit model which accounts for this phenomenon.

Another source of asymmetry arises from the fact that messages can span across multiple rendezvous times. Thus a node that has selected an earlier time can block one that has selected a later one (up to cyclic permutation), but not vice versa[1]. The notion of partial conflict arises again: the distribution of message sizes may be such that long messages cause a conflict, but not short ones.

3.3.2 Conflict ratio

Let us define the *conflict fraction* f as well as some relevant state variables:

$$\begin{aligned} f &= \text{fraction of messages received by a node that} \\ &\quad \text{were destined for another node} \\ N &= \text{the (estimated) number of neighbors of a node} \\ M &= \text{the time to send an average message} \end{aligned}$$

[1]This problem does not occur if the two nodes use different frequencies or spreading codes; using multiple frequencies makes conflict less likely and is thus unequivocally desirable.

s = the number of rendezvous times per period

f = the number of different frequencies available

First, let us note that f captures the *observed* amount of partial conflict; this operational definition includes conflict due to hidden terminals, long-distance radio interference, and packet length variations. In fact, all we care about is this definition, since it does not really matter *why* messages are getting through. Accordingly, if f is high, a node A should want to switch channels more. At the same time, switching causes a problem for all of A's children, which must detect A's absence and switch parents or wait until they discover A's new channel.

It is important to note that it is only worth switching if things will be better after the switch. Therefore, we calculate the *expected conflict fraction* e as follows. First note that if we assume that each node has the same number of children, we need only calculate the number of nodes that have a sufficiently "close" channel to A that a message to that node could interfere with one destined for A (in other words, we don't need to weight them by the frequency of sending). In this case, "close" means that both nodes' channels have the same frequency, and that the other node's rendezvous time is within one message time of A's rendezvous time. Thus we have

$$CLOSE = \frac{N}{f}\frac{M}{T}$$

In other words, the number of close nodes is the number of nodes on the same frequency (assume uniformly random choices), times the fraction of the period taken up by a message. The latter term comes from the fact that it is equal to the probability that A's rendezvous time lies inside the extent of a transmitted message. Then we simply have:

$$e = \frac{CLOSE}{1 + CLOSE}$$

To combine the cost and benefit, Let us define the *switching pressure* p:

$$p(f, c) = \frac{f - e}{g(c)}$$

The function $g(c)$ is a weighting function on c, satisfying the properties that $0 \leq p(f, c) \geq 1$, which implies that $g(c) \geq 1$; in order to make sense as a cost function, $g(c)$ should also increase monotonically. Since children generally have a node to failover to (see Section 3.4), the cost of switching is not too large, and $g(c)$ should reflect this fact.

We compute f as follows. First, we keep a sliding window of the status of last m packets received by a node, assigning a value to each packet. We assign a message 0 if it is destined for the node that receives it; α if a message in progress was detected using carrier sense; and some fraction β of the total message time for messages that start at the right time but are destined for another node. The value α is a system-specific parameter that measures what fraction of a message time is required to detect a carrier. The value β is the fraction of a message that needs to be looked at to decide who the correct recipient is. We then calculate f by taking the mean of these samples over the sliding window. We may think of f as describing the recently observed conflict ratio on the channel; note that for suitable choices of α and β, we can store each sample using only a few bits. The value of c is just equal to the number of samples with value 0; if the function $g(c)$ is compute-intensive, it may be precomputed into a table (since the number of possible values is bounded by the size of the window). Determining the neighborhood size may be done via implicit sampling of the announcement channel (see Section 3.4) to determine how many nodes are present. It is straightforward to compute p at this point.

3.3.3 A solution

We draw inspiration for our solution from the the distributed coloring algorithm of Fitzpatrick and Meertens[MF01]. Their Conservative Fixed Probability (CFP) algorithm works as follows (applied to the current problem): each node independently chooses a channel at random, then if a conflict is detected, the node switches channels with some fixed probability. This very simple approach is shown to converge relatively rapidly. Since the coloring does not change unless there are conflicts, and the probabilistic approach (with a sufficiently low probability) ensures that not all conflicting nodes will switch, it is stable as well.

As we noted above, our problem is not quite graph coloring, since we need to deal with partial conflicts. Nevertheless, we use CFP's idea of nodes making local choices and resolving conflicts as they occur. A key change is that our system uses a variable probability that depends on the recent observed behavior of the channel.

What is that probability? The switching pressure provides a reasonable value: it is a measure of the probability of seeing a net benefit from switching channels. It is also likely to be very small in the steady state, so the solution will be stable. Thus with probability p, a node switches to different channel, randomly chosen from among channels not known to be allocated (based on its neighbors' announcements).

3.4 Node State Exchange (Announcements)

We have discussed how a node decides whether or not to choose a new channel on which it listen for messages from its children. Having chosen a channel, though, it must convey that choice to nodes in its neighborhood that might want to forward packets to it.

The goals of state exchange are, first, to do it as little as possible when the network is not changing (with some exceptions; see section 5.2), and second, to disseminate new information quickly when the network is changing. There are two components of the exchange: listening for announcements and sending announcements on a special well-known channel we call the *announcement channel*. The former benefits the node doing the listening, as it may discover a better route; the latter benefits the node's (potential) children.

The problem is similar to that of neighbor discovery (as that is one of its functions). One approach to this problem is to use "Birthday Protocols" (BP)[MB01], which exploit the birthday paradox to find neighbors without requiring a lot of sending or listening time. The idea is to listen during each time slot with some probability p_l, send with probability p_s, and do nothing otherwise. The birthday paradox makes it likely that even for low values of p_l and p_s, two nodes will hear about each other. They explicitly analyze the case of an N-clique, and show that the "Probabilistic Round Robin" (PRR) mode works well for link discovery.

We have somewhat different goals and assumptions. First, our goal is connectivity, not maximum link discovery. As long as the graph is connected, it doesn't matter how many links are undiscovered. Additionally, unlike in the BP analysis, we do not assume that there is a fixed initial phase of state exchange; it must be an ongoing process to account for changes in the network. Lastly, the BP analysis calculated a suitable p_l in terms of the desired fraction of links discovered in certain time. Our target is to say that each node should have heard from some number m of potential parents in the last T seconds, so that connectivity is verified and failover to a new parent is fast and reliable. We call this the *recency constraint*.

This boils down to an optimization process. Let us define an input parameter $H = \frac{m}{T}$, which defines the average number m of parent announcements we wish to hear in an interval T. Let M_i be the set of potential parents in range of node i, N_i be the set of all other neighbors in range of i, and c_s and c_l be the energy cost of sending and listening, respectively.

Recency constraints:

$$\forall i \, \Pr[\text{node i is listening}] * \Pr[\text{exactly one parent node of i is sending}] = H$$

Condition	Action
None	Decay p_s by a factor of β
Heard announcement from parent-less node	Set p_s to p_s^{max}
Observed collision on announcment channel	Set collision bit in next announcement
Heard announcement with collision bit set	Decay p_s by a factor of $1 - 1/N$.

Figure 3: The probability p_s of sending an announcement during the next announcement period is adjusted depending on recent events. The decay parameter for p_s is β, and p_s^{max} is the maximum send probability, set equal to $1/2N$.

$$\forall i(p_l^i) \sum_{j \in M_i} \left[p_s^j \prod_{k \in M_i - j} (1 - p_s^k) \prod_{l \in N_i} (1 - p_s^l) \right] = H$$

Objective function (minimize energy):

$$\sum_i (p_s c_s) + (p_l c_l) - (p_s p_l)(c_s + c_l)$$

Unfortunately, this is a complex nonlinear problem, so it is unlikely that even a centralized approach would yield an exact solution on the time scales we require. Fortunately, each node can evaluate its relevant constraint locally (by remembering a short history of possible-parent announcements). So if each node attempts to minimize its energy consumption while maintaining the recency constraint, we can calculate a solution in a local, distributed fashion.

Our basic proposal is that both send and listen probabilities should be decayed over time unless an event occurs that requires increasing them. Each period, p_l and p_s will be decayed by the *decay parameter* β, so the net effect is an exponential decay. Thus, we expect that in the steady state, energy spent on these maintenance messages will be very low.

3.4.1 State exchange listen probability

Setting the listen probability is now relatively straightforward: a node's listen probability should be increased as needed to maintain the recency constraint. A simple algorithm is to raise the listen probability to 1 when the constraint is violated (for example, when a node first joins the network or its parent has died and it has no backup). When sends and listens conflict (which they will if the listen probability is set to 1) then sends take priority.

3.4.2 Announcement send probability

Next, we deal with send probabilities. We feel that in the BP analysis, send conflicts were not sufficiently accounted for. When a conflict occurs—two nodes send announcements heard by some destination—both packets may be lost and much energy is wasted sending and listening. The probability of collision in the N-clique is

$$\begin{aligned} P_c &= 1 - \Pr[\text{no node sends}] - \Pr[\text{exactly one node sends}] \\ &= 1 - (1 - p_s)^N - (N)(p_s)(1 - p_s)^(N - 1) \end{aligned}$$

Say $p_s = \frac{1}{\alpha N}$. Then

$$P_c = 1 - (1 - \frac{1}{\alpha N})^N + (N)(\frac{1}{\alpha N})(1 - \frac{1}{\alpha N})^N$$

Approximating using $N \approx N - 1$ and using the fact that $N = \frac{\alpha N}{\alpha}$),

$$\begin{aligned} P_c &\approx 1 - (1 - \frac{1}{\alpha N})^{\frac{\alpha N}{\alpha}}) - (N)(\frac{1}{\alpha N})(1 - \frac{1}{\alpha N})^{\frac{\alpha N}{\alpha}}) \\ &\approx 1 - \exp^{\frac{-1}{\alpha}} - \frac{1}{\alpha} \exp^{\frac{-1}{\alpha}} \\ &\approx 1 - (1 + \frac{1}{\alpha}) \exp^{\frac{-1}{\alpha}} \end{aligned}$$

Thus, the suggestion that p_s should in general be $1/N$, where N is the estimated number of neighbors, is probably too high, since the expected probability of conflict is $1 - \frac{2}{e} \approx 0.26$, which means that about 1 in 4 announcement slots there will be a conflict and the energy used by all senders and listeners. We can lower this probability to about 1/10 by using $p_s = 1/2N$.

Thus we propose that p_s be capped at a maximum of $1/2N$. The only cost to this is that a desperate listener will have to wait one additional slot on average, which seems like a small price to pay (not to mention that all senders are sending half as often).

The send rate should be increased to the maximum when an announcement is heard from a node with no parent, since this is the most urgent case, or if it changes levels (information which should be propagated). The send rate should be set to $1 - 1/N$ of its old value if a node hears an announcment with a special "collision" bit set. This bit will be set by a node sending an announcement if it has heard a collision on the announcement channel since its last announcemnt. The factor $1 - 1/N$ is designed so that of the N neighbors of the node that noticed the collision do not all back off too much. The assumption here is that the N's for each node in question are either similar or distributed in such a way that the aggregate send rate is reduced appropriately.

We conclude our discussion with the note that although we have defined our probabilities in terms of the probability of sending during a particular announcement period, in fact all we want is to know the next time to listen or send. We may want to turn off the CPU during unneeded periods. Fortunately, one can get the same result as activating with probability p during each period by choosing interactivation times by sampling from an exponential distribution with mean $1/p$.

3.5 Switching Parents

We have so far only talked about a good strategy for optimizing the receiving portion of communication, i.e., selecting and switching channels. The complementary—and easier—problem is that of a sender's knowing when to switch parents. Work by Woo and Culler[WC03] shows that a Window-Mean EWMA is both stable and responsive as a link estimator in the sensor environment. We apply this not merely to measure the radio link, but to measure the empirical *usefulness* of a link, including the effects of schedule conflicts. In short, a node attempts to switch parents when, for whatever reason, its messages are not getting through to its parent with very high probability. A new parent is selected from among those in its parent table, build using announcements it has heard.

3.6 Individual Node Behavior

For the present discussion, we assume tree-based routing. Nodes may send packets containing aggregated sensor readings to any node that is one hop closer to the base station, i.e. any node in its parent set. How exactly a node chooses the "best" parent we do not analyze in this paper; metrics for fairness are discussed in Section 5.2.

Each node simply transmits its packet on its parent's advertised channel. Acknowledgements are done at the link layer and may be aggregated across multiple packets. If a message is not sent due to carrier sense

or, if enabled, collision detection, exponential backoff is performed, with some maximum backoff before the parent node is switched and the packet discarded. One key difference from a normal CSMA MAC layer is that backoffs are not done in arbitrary units of time; they are done in units of periods. Thus a node always transmits on its parent's channel, even when it back off.

A node always listens on its receive channel when it has promised to do so. If a message is heard, it stays on for as long as it takes to complete the message, even if it spans multiple slots. The default listen rate is once per period; please see Section 5.1 for an adaptive approach.

Channel allocation and state exchange proceed according to the algorithms in the previous sections.

4 Simulation

Note: The simulation and results presented here are based on a slightly older version of STEAM than the one described here; however, the basic results are valid.

4.1 Setup

In order to test the correctness and efficacy of our scheme, we developed a simulation of a wireless sensor network. It is a packet-, not bit-, level simulation, but does take into account the hidden terminal problem, where a new message can corrupt one currently in progress. Simulation was by time step, where each step was the size of one slot in the TDMA scheme. Radio range was assumed to be a perfect sphere, with no falloff within the sphere, and perfect falloff outside. We did not explicitly simulate link-layer ACKs; however, they were accounted for: nodes were made aware of which messages got through to the next hop and which did not. Likewise, although our simulator was capable of handling probabilistic packet loss, such loss would only contribute to the setting of the input parameters that control the degree of hysteresis, so we do not include results with explicit packet loss. Loss due to collisions or carrier sense, however, were accounted for. The fixed parameters are shown in Figure 4. Radio parameters were derived from the ChipCon CC1000 radio[CC1000] used in the latest Berkeley nodes. Message-related parameters were derived from typical sizes in other sensor network research.

50 nodes were distributed randomly in a circle of radius 15 units; the transmission radius was 5 units. Interestingly, this resulted in some nodes being 7 hops from the base station!

Initial parent selection for a node was done by a random choice from among all possible parents of that node.

Four events take place during the simulation. One node is added 2 seconds in, 2 more are added 20 seconds in, and one of those is "killed" at 50 seconds in. These changes are to force at least part of the network to adapt. It may be helpful to look at the diagrams for the network at different points: see Figures 5,6,7, and 8. Solid lines indicate a parent; dashed line indicate a potential parent.

4.2 Channel Allocation

The fundamental goal of channel allocation is to minimize power while maintaining connectivity (a high fraction of successfully transmitted packets). One control parameter for our channel switching algorithm is the sliding window size for received messages. A larger window will give you a more stable network that can handle transient problems, but will be slower to react to change. In our experiments, this parameter did not seem to make a big difference when it was 10 packets or more. We thus used a 10-packet window. We also fix the H parameter at 5 (so parents will be heard every 5 seconds on average)..

To evaluate our channel-switching strategy, we tried to compare the channel-constrained performance of the algorithm against the ideal case where each node can select a unique channel. Our metrics were energy consumption and packet loss rates. Unfortunately, in our tests, we could not find an "interesting" part of

Slots per period	10
Periods per second	10
Number of nodes	50
Transmission speed	40 Kbps
Send power	10 (mA @ 3V)
Receive power	17 (mA @ 3V)
Packet size (bytes)	30
Maximum backoff on carrier sense (periods)	8
Slots per announcement packet	1
Decay rate for announcement send/listen probability	0.95
Minimum probability of listening for announcements	0.1
β parameter from Section 3.3	0.5
Consecutive packet losses to switch parents	3
Sensor reading interval	5 sec

Figure 4: Fixed parameters in the simulation.

the overall parameter space (there are some 20 parameters). In other words, with a reasonable number of channels (as few as 7 slots with 1 code), there were virtually no dropped packets (4% or so, largely due to two children sending at the same time) even with the test nodes entering and leaving.

4.3 State Exchange

The primary "control knob" for our state exchange algorithm is the rate at which you expect to hear parent announcements. This number is significant since it provides a temporal guarantee for connectivity in the network. Essentially you are trading off lower power consumption for slower response to changes in the network that require route modification.

There are two things to check: first, that the parameter indeed corresponds to the time it takes for new information to propagate; and second, how the input parameter affects the tradeoff between the fraction of packets successfully delivered in the face of a changing network and the amout of power consumed doing so.

As to the former, our experiments showed that convergence (an optimal network by a distance vector metric) was achieved for any reasonable value of the parameter, and the convergence time was roughly proportional to it.

5 Extensions

5.1 Adaptive Listen Rate

While nodes ordinarily always listen once per period, this scheme does not take into account observed levels of traffic, which may suggest listening more or less often. The latter case is more common, since we expect period will be sufficiently short that needing to receive multiple messages per period will be rare. Listening less often will improve energy consumption at the cost of longer delays. This tradeoff may acceptable for many applications, so we propose the following extension.

An EWMA estimator is maintained to estimate the incoming traffic rate. Each time a node wakes up it records a sample of 0 or 1 according to whether or not it received a message for itself. Then we choose upper and lower thresholds such that if the estimator exceeds the upper threshold, the rate is increased and vice-versa for the lower threshold. The thresholds are needed to maintain a degree of stability with respect

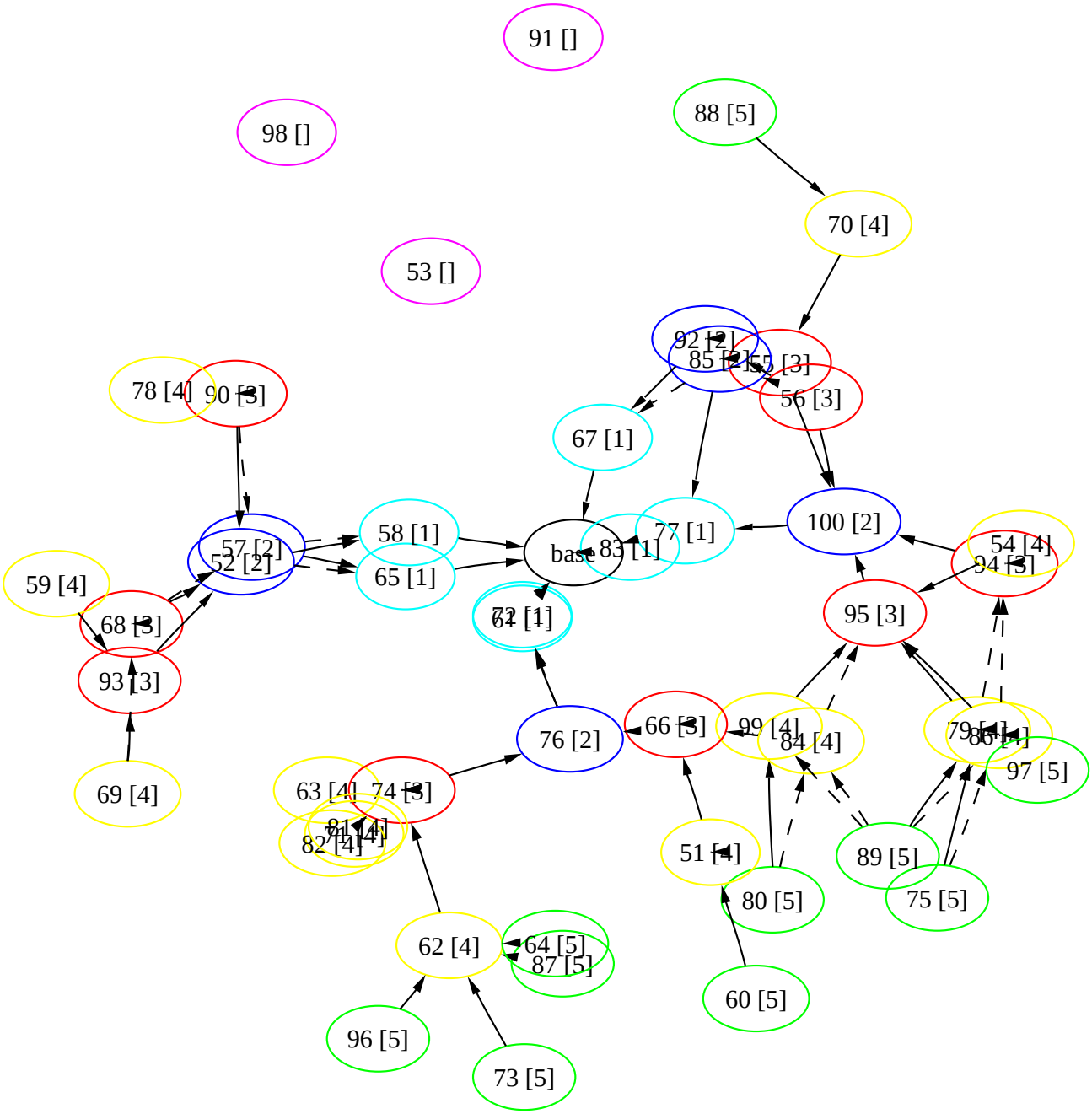


Figure 5: The initial state of the network. The numbers in brackets are the number of hops from the base station. This is also indicated by the color of the nodes (if available). Solid lines indicate a parent; dashed line indicate a potential parent.

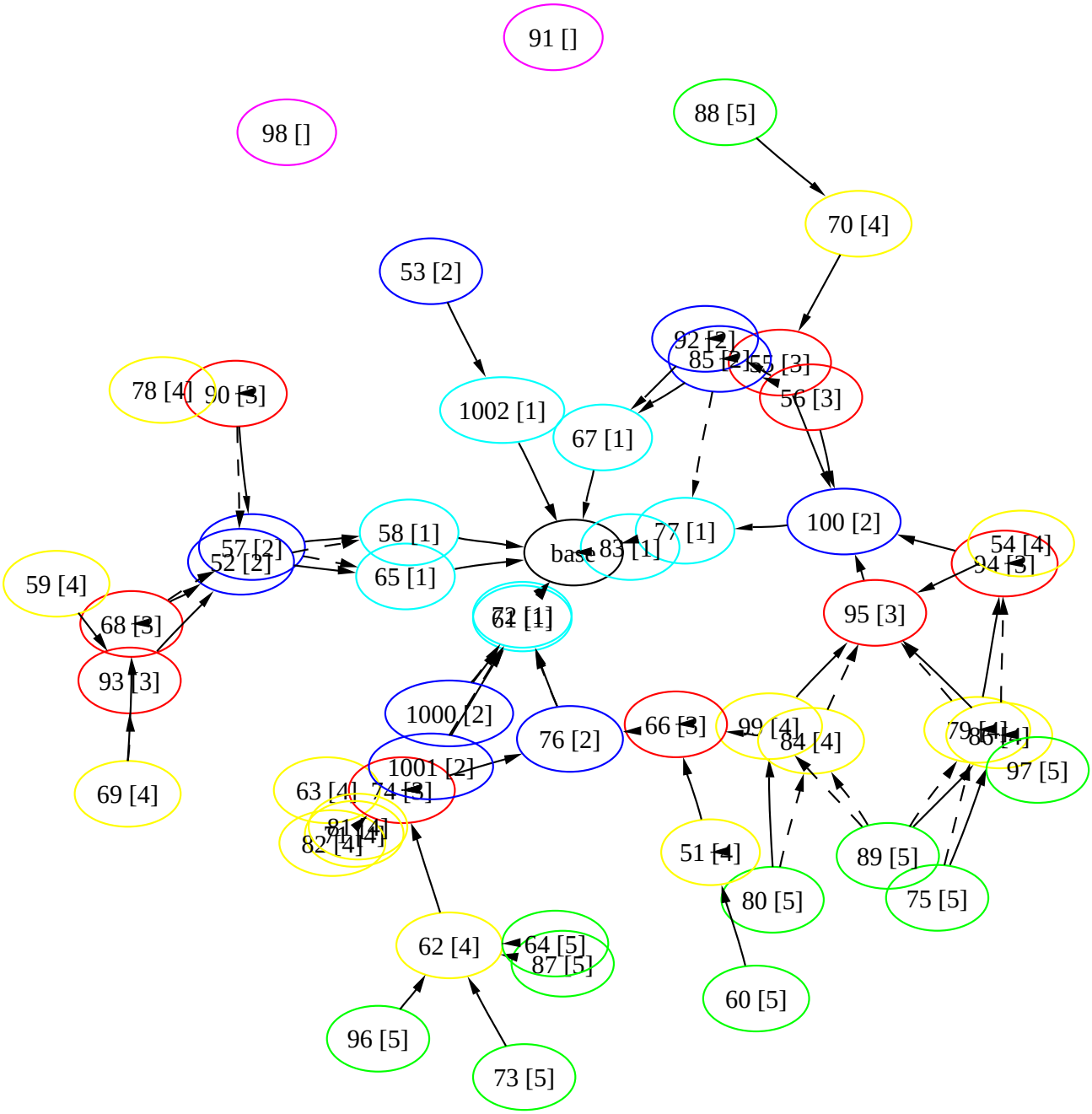


Figure 6: At 20 seconds in, three more nodes (1000, 1001, 1002) have been added, but the network has not yet adapted to 1000 and 1001, which just arrived. 1002 is already routing packets for the formerly partitioned node 53; node 1002 arrived 2 seconds in.

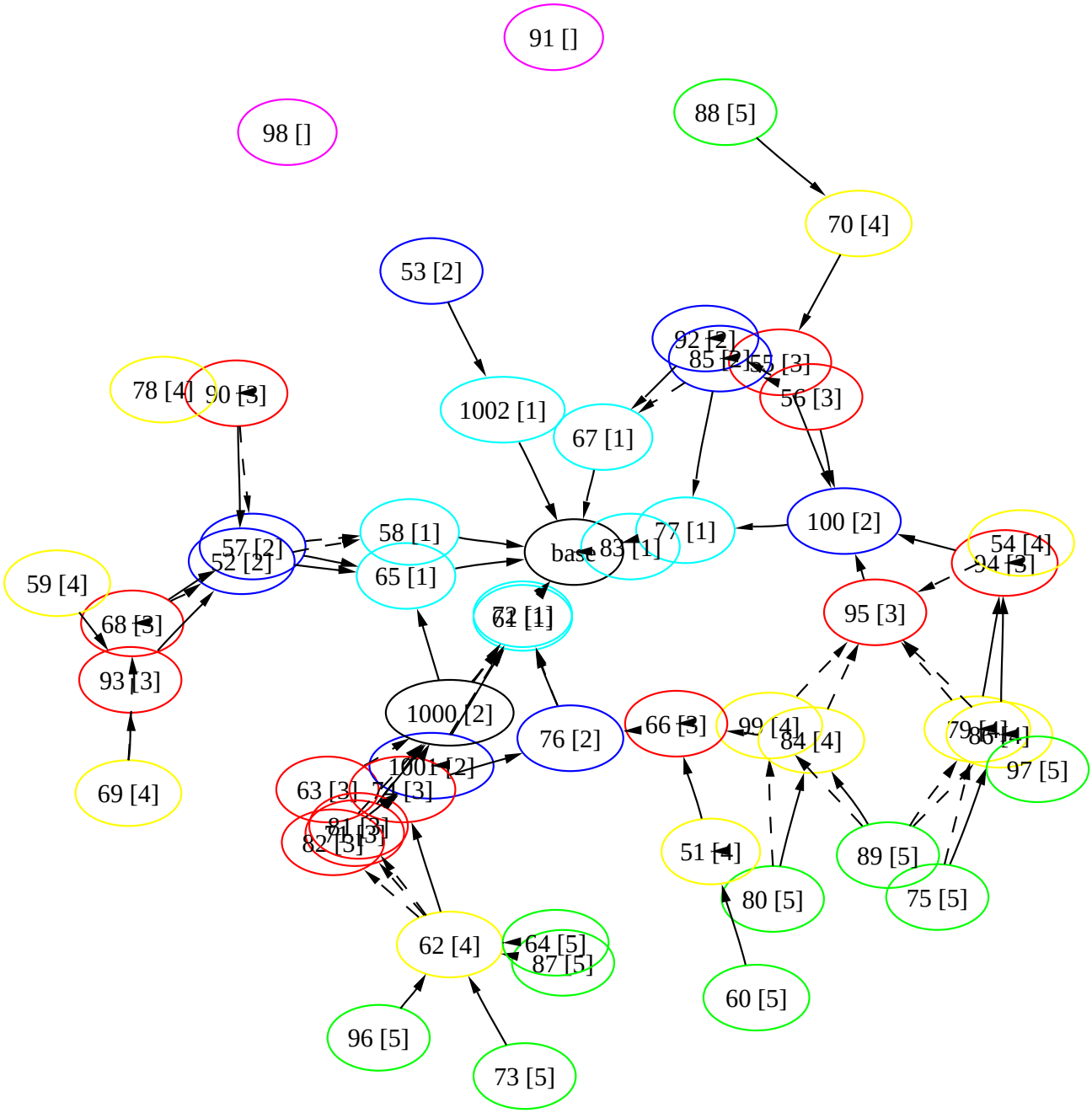


Figure 7: At 50 seconds in, node 1000 is killed (it just stops responding). Notice, though, how the cluster of nodes below and to the left of 1000 and 1001 has already adapted by routing through the new nodes, which eliminated one hop.

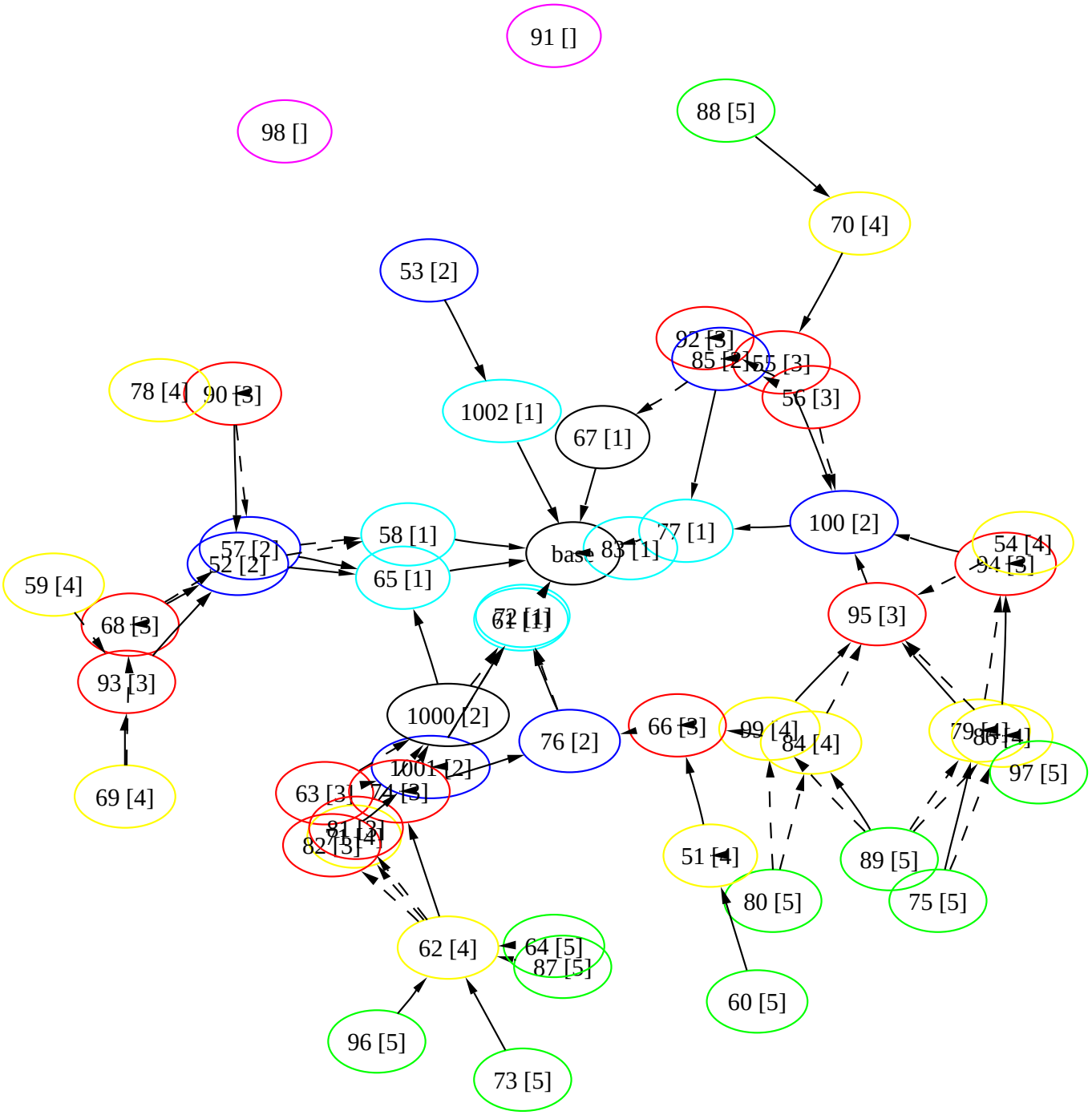


Figure 8: At 150 seconds in, no more changes are taking place. The network has adapted to 1000's death. Nodes which were routing through it have switched parents.

to local peaks and valleys in traffic levels. Naturally, when a node changes its listen rate from, say, every period to once every three periods, it must announce this fact on the announcement channel, and ideally, inform each child node of the decision when it receives a packet. To satisfy the latter request, each node that decreases its listen rate should respect the old rate for long enough that each child has sent one packet to it (and thus has been notified of the change).

5.2 Adding Fairness

One difficult to avoid issue it that some nodes have to route more packets than others. Some of this is unavoidable: edge nodes don't have as much work since they don't need to route anything. With a small diameter network, "a lot" of nodes are on the edge, e.g. for 5 hops, 9/25 ($> 1/3$) are in the outermost hop assuming a uniform distribution circularly around the base station.

If nodes are careful about choosing parents, network lifetime can be extended, especially if some nodes have longer battery life. We need a metric whereby each node can decide which possible parent to send to. The metric is going to be some function of the target's battery level, the target's current forwarding rate, and the energy required of the sender to reach the target. These values are gleaned from periodic state exchange messages that each node will send out.

6 Related Work

There have many been many efforts in the area of energy-aware communications recently. There are roughly three classes of schemes: TinyOS' Low-power listening[], which operates at a hardware level; drop-in MAC layer replacements, including S-MAC[YHE02] and T-MAC[VL03]; and explicit scheduling approaches, including TRAMA[ROG03], Flexible Power Scheduling[HDB03], and DuraNet[MTW03].

6.1 Existing Approaches

TRAMA The Traffic-Adaptive Media Access (TRAMA) protocol is an explicitly scheduled system. Nodes must maintain consistent two-hop topology information, along with the schedules of every node in this radius. Using the information, the nodes can figure out when it is safe to send without causing a collision. Latency can also be high since the schedule period is sufficiently large that packets are more frequently queued at each node. It is not known how easy it is to maintain two-hop-radius information in practice, especially in the face of faulty nodes and changing topology. TRAMA attempts to fail in such a way as to preserve correctness; it is not clear to what extent a real network would engage this mode, which has high energy consumption.

Flexible Power Scheduling The Flexible Power Scheduling (FPS) [] approach of Hohlt et al. takes the approach that traffic consists of flows and that bandwidth should be reserved for these flows from each node to the base station. This is achieved using a supply-and-demand model. Each node demands a certain amount of bandwidth according to the traffic it generates and the traffic it must forward from its descendants in the routing tree. The node's parent supplies the bandwidth by making reservations for each unit of bandwidth to be sent during some particular time slot in each period. The time slot is long enough that if there is interference with some other pair of nodes, a standard CDMA MAC can resolve it. The benefit of this approach is that end-to-end yield is good, since enough bandwidth is reserved all the way to the root of the tree. Peaks in traffic are handled by buffering at the source nodes (since they must respect their reservations); this means that, first, reservations must be made for the expected average sending rate and that, second, latency during peak times can be high. This could be problematic since each period is on the order of seconds. Compared with STEAM, FPS reduces the precision of required time synchronization

and may have somewhat better steady-state performance at the cost of long latency and the need for queues during peak periods.

DuraNet DuraNet attacks the reduced problem of ad-hoc networks with relatively stable connectivity. The idea is to set up a fixed pairwise schedule that ensures no conflicts. During an initial setup phase, RTS/CTS is used cleverly to schedule times for future communications. When a pair of nodes can successfully exchange an RTS/CTS, they know that that offset into the period is free for communication during each subsequent round of communication. DuraNet has the advantage of being almost perfectly efficient, since the sender and receiver always agree and have no conflicts with other nodes. The disadvantage of the scheme is that it is not particularly robust to changes in the interference topology or joins and leaves. Thus it is likely to be well suited to relatively static monitoring applications.

TinyOS The TinyOS *low-power listening mode* makes a physical-layer adjustment as follows:

1. Choose a particular duty cycle D. The duty cycle basically is the fraction of time the radio may be turned on (typically for listening; the send rate is usually fixed by the sensor sampling rate). For our example, let us $D = 0.01$, or 1%.

2. Turn on the radio for 1% of the time, say for 2 ms every 200 ms.

3. During this interval, listen for a special *start symbol* being broadcast. If you hear one, keep the radio on and wait for the actual message. If not, turn the radio off.

4. If you are a sender, send a start symbol for (in this case) 200 ms, then send the actual message. It is necessary to send for the full 200 ms—in general, for a time inversely proportional to D—so that the intended receiver will be awake.

This scheme, while simple and requiring no extra coordination, has two undesirable properties. First, as D *decreases*, senders and listeners must stay on send for an *increasing* amount of time. Second, listeners must listen to messages (and, more importantly, long start symbols) that are not destined for them. This is wasted energy. Low-power listening is most effective when the message rate is very low. In this case, steady-state energy consumption is what is most important, and low-power listening allows us achieve a low duty cycle transparently and without any overhead.

S-MAC S-MAC is a drop-in MAC layer replacement for a default CSMA MAC. Its efficacy is based on the simple principle that if every node buffers its outgoing messages, then all the nodes send their buffered messages during a short, intense burst, then the nodes can turn off their radios between bursts. It uses fixed on periods and fixed off periods. Synchronizing these periods—maintaining *virtual clusters*— can be tricky, and in some cases nodes may have to adopt multiple schedules. The scheme also requires enough message buffer space to store all queued messages.

T-MAC T-MAC is an improvement over S-MAC in that it uses variable-length on- periods and certain other optimizations that attempt to reduce problems such as early sleeping and send-priority for nodes with full buffers. While good results were achieved in simulation of static networks, it is not clear how well the protocol performs in more dynamic situations with joins, leaves, and changing interference.

Other Related Work Xu et al.[XHE00] consider a neighborhood-adaptive scheme, but it does not have multiple channels and thus achieves duty cycles on the order of 50% rather than a few percent.

We have discussed the CFP coloring algorithm[MF01] in Section 3.3; as we have noted, though, our problem here is not in fact graph coloring, though our solution certainly owes a debt to its local, dynamic approach.

Birthday protocols[MB01] (see Section 3.4) are an efficient solution to neighbor discovery. Our metric is different (temporal guarantees of connectivity), but again, our algorithm builds on this approach.

Ramanathan and Rosales-Hain[RR00] explore a method of topology control by adjusting the radio send power and sensitivity. We feel this is an important area to consider when dealing with overloaded nodes; we plan to investigate this approach within our framework.

7 Conclusion

We have presented a new system of communication in sensor networks that combines a TDMA MAC with a simple, adaptive method of routing packets that achieves low overhead without sacrificing connectivity. Our algorithms were designed to be distributed, iterative approximations to theoretical optima. Our simulation shows that correctness is achieved, although we do not have tight bounds on how fast or within what factor of optimal power our algorithm performs.

There are some other open questions: how important is fairness? Do you want fewer nodes to live longer, perhaps maintaining a single connected tree, or for all the nodes to die at the same time?

We have only begun to explore the huge parameter space of our system. There is a lot of room for further analysis, but we feel that our framework gives us a good starting point for achieving efficient low-power operation with realistic assumptions.

Acknowledgments

Many thanks go to Robert Johnson for his help on the theoretical analysis of STEAM. We would also like to thank Ion Stoica and David Wagner for helping us focus on the important issues in this design space.

References

[CC1000] Chipcon CC1000 RF Transceiver Data Sheet. Available at http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf.

[EE01] J. Elson and D. Estrin. “Time Synchronization for Wireless Sensor Networks”. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, April 2001, San Francisco, CA, USA, pp. 1965–1970.

[HC02] J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. In *IEEE Micro*, 22(6):12-24, Nov/Dec 2002.

[HDB03] B. Hohlt, L. Doherty, and E. Brewer. “Flexible Power Scheduling for Sensor Networks.” UC Berkeley Technical Report UCB//CSD-03-1293. Berkeley, CA, January 2003.

[MB01] M. J. McGlynn and S. A. Borbash. “Birthday Protocols for Low Energy Deployment and Flexible Neighbor Discovery in Ad Hoc Wireles Networks”. In *Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking and Computing*.

[MF01] L. Meertens and S. Fitzpatrick. “Soft, Real-Time, Distributed Graph Coloring using Decentralized, Synchronous, Stochastic, Iterative-Repair, Anytime Algorithms: A Framework”. Kestrel Institute Technical Report KES.U.01.05, May 2001.

[MTW03] D. Molnar, T. Tong, and A. Woo. “DuraNet: Energy-Efficient Durable Slot-Free Power Scheduling”. UC Berkeley CS262A class project report.

[ROG03] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. “Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks”. In *Proceedings of SenSys ’03*, Los Angeles, November 2003.

[RR00] S. Ramanathan and R. Rosales-Hain. “Topology Control of Multihop Radio Networks Using Transmit Power Adjustment”. In *Proceedings of IEEE Infocom 2000*, Tel Aviv, March 2000.

[VL03] T. van Dam and K. Langendoen. “An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks”. In *Proceedings of SenSys ’03*, Los Angeles, November 2003.

[WC03] A. Woo and D. Culler. “Evaluation of Efficient Link Reliability Estimators for Low-Power Wireless Networks”. UC Berkeley Technical Report, November 2002.

[XHE00] Y. Xu, J. Heidemann, and D. Estrin. “Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks”. Research Report 527, USC/Information Sciences Institute, October, 2000.

[YHE02] W. Ye, J. Heidemann, and D. Estrin. “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 2002)*, New York, NY, USA, June, 2002.