# Embedded Networks:

## Pervasive, Low-Power, Wireless Connectivity

**Robert Dunbar Poor**

Submitted to the Program in Media Arts and Sciences School of Architecture and Planning in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the Massachusetts Institute of Technology

January 2001



**Author**

Program in Media Arts and Sciences
December 1, 2000

**Certified by**

**Michael J. Hawley**
Assistant Professor of Media Arts and Sciences
Thesis Advisor

**Accepted by**

**Stephen A. Benton**
Chair
Departmental Committee on Graduate Students
Program in Media Arts and Sciences

## *Abstract*

The lack of effective networking technologies for embedded microcontrollers is inhibiting the emergence of smart objects and "Things That Think."

A practical communication infrastructure for Things That Think will require wireless network connections built directly into microcontroller chips. After showing that digital processing, application languages, and wireless links are not the bottleneck, this thesis turns its attention to network designs. It presents architectures and algorithms that implement self-organizing networks, requiring minimal pre-planning and maintenance.

The result is a radically new model for networks—*embedded networks*—designed specifically to interconnect untethered embedded microcontrollers. The thesis culminates in the design, implementation and evaluation of a hardware system that tests and validates the approach.

Thesis Advisor:
**Michael J. Hawley**
Assistant Professor of Media Arts & Sciences

This research was sponsored by the Things That Think Consortium. The author gratefully thanks the Motorola Fellows Program, the AT&T Fellows Program and DARPA for their support.

# Embedded Networks:

## Pervasive, Low-Power, Wireless Connectivity

**Robert Dunbar Poor**

*The following people have served as readers for this thesis:*

**Reader**

**Andrew Lippman**
Senior Research Scientist
Media Laboratory
Massachusetts Institute of Technology

**Reader**

**William J. Kaiser**
Professor
Electrical Engineering Department
University of California, Los Angeles

# *Contents*

# *List of Figures*

# CHAPTER 1 *A Network on Every Chip*

> *A trillion dumb chips connected into a hive mind is the hardware. The software that runs through it is the Network Economy. A planet of hyperlinked chips emits a ceaseless flow of small messages, cascading into the most nimble waves of sensibility. Every farm moisture sensor shoots up data, every weather satellite beams down digitized images, every cash register spits out bit streams, every hospital monitor trickles out numbers, every Web site tallies attention, every vehicle transmits its location code; all of this is sent swirling into the web. That tide of signals is the net.*

—Kevin Kelly "*New Rules for the New Economy*" [Kelly 1997]

## *An unfulfilled promise*

For years, visionaries have predicted that tiny computers will soon be woven into the everyday fabric of our lives and a world densely populated with "smart objects," giving rise to "Ubiquitous Computing," [Weiser 1991], "The Network Economy" [Kelly 1997], and "Things That Think" [Gershenfeld 1999]. These predictions have not yet been realized. Why not?

Processing power has become cheap and plentiful. Dollar for dollar, microcontrollers are a thousand times faster than a decade ago [Moravec 1998]. In the year 2000 alone, the total production of microcontrollers exceeded the world population [Tennenhouse 2000]. These tiny chips are being embedded into everyday objects—watches, pacemakers, smart cards, traffic lights, children's toys—at a prodigious rate. Clearly, available processing power is not the limiting factor.

Languages for microcontrollers have also proliferated. Mobile agents [Minar 1999], "thin clients" [emWare 2000], JINI [Sun 2000] and dozens of other computationally lightweight languages have been developed to support dedicated applica-

tions in embedded devices. Availability of these languages has not resulted in the predicted explosion of smart objects.

The steadily falling price of microcontrollers has resulted in situations where the cost of a single connector can exceed the cost of the microcontroller it connects[1]. In the last few years, industry standards such as IrDA [IrDA 1998], IEEE 802.11 [IEEE 1999], and Bluetooth [Bluetooth 1999] have created wireless interconnect systems that are less expensive than their wired counterparts. Since these wireless technologies themselves make heavy use of semiconductor technologies, they enjoy progressively lower cost and increased communication rates per unit power.

Despite the availability of these essential ingredients—cheap, abundant processing; lithe application languages; and inexpensive wireless links—few everyday objects show any signs of increased intelligence.

## *Networking: the missing link*

A typical embedded microcontroller works in relative isolation, unable to draw upon information or exert any influence beyond its immediate realm. For all its computing power, it is like a genius sequestered in a basement: smart and capable, but having neither sensory inputs to give it context nor the means to express what it knows. We are left with ubiquitous but senseless computing and billions of Things That Think which cannot relate.

*Legacy networks are ill-suited for linking embedded microcontrollers.*

Talk is cheap, at least among humans. But for the tiny embedded microcontrollers found in common objects, the cost of discourse remains relatively high. Today's digital networks were originally designed to interconnect mainframe and minicomputers and have been adapted, somewhat awkwardly, to connect PCs and lap-

1. A spot check of a popular electronics part supplier shows that in quantities of one hundred, the popular DB9 serial connector costs $3.12, a microcontroller that processes one million instructions per second costs only $0.94.

top computers. These legacy networks are ill-suited for embedded processors: they cost too much, they consume too much power, and they don't scale well to handle the hundreds and thousands of connections required in a world of Things That Think.

## *Embedded Networking*

The lack of effective networking technologies for embedded microcontrollers is inhibiting the emergence of smart objects. What is required is a new model of networking—*embedded networking*—designed specifically to interconnect embedded microcontrollers.

A network must attain a critical mass if it is to be useful. As proposed by "Metcalfe's Law," the value of a network rises as the square of the number of devices connected. In a world where the number of embedded microcontrollers is growing exponentially, the only reliable way to arrive at and to maintain critical mass is to put the network connection directly on the chip.

## *The domain of Embedded Networking*

Herb Simon points out that it is useful to consider a technology as *"an 'interface'... between an 'inner' environment, the substance and organization of the artifact itself, and an 'outer' environment, the surroundings in which it operates."* [Simon 1969]. Embedded Networks are built into embedded processors and provide communication links for specific applications in a relationship portrayed below in Figure 1. These contexts dictate the fundamental design requirements of embedded networks.

*Embedded Networking is implemented on microcontrollers (its "inner environment") and interacts with dedicated applications (its "outer environment"). Each environment dictates constraints upon its design.*

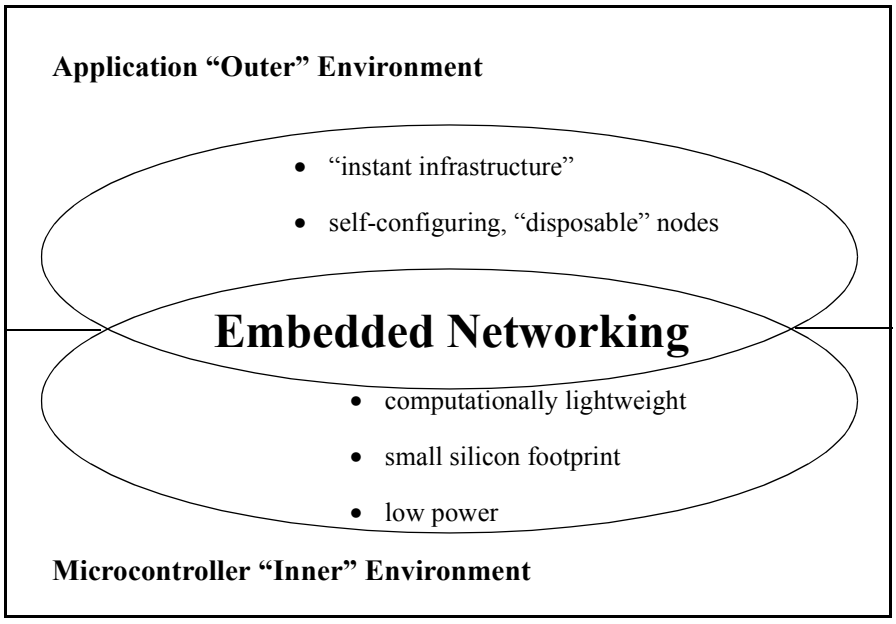



**FIGURE 1. Context and constraints of embedded networking**

## *Constraints imposed by the host*

*An Embedded Network node must not overly tax the microcontroller chip on which it is built.*

The host microcontroller on which the Embedded Network node is fabricated—its "inner environment"—imposes a set of constraints. The power of microcontrollers lies in their generality: a single microcontroller architecture is suitable for a broad range of applications. For embedded networking to be viable, it too must be adaptable to a broad range of applications. Since the embedded network system resides on the microcontroller chip itself, it must not impose a significant burden on the chip, giving rise to the following design principles:

### LOW POWER CONSUMPTION

For an embedded network node to be an attractive candidate for integration onto a microcontroller, it should not exceed the power consumption of the microcontroller itself.

One of the consequences of Moore's Law—the proposition that the number of transistors per unit area of integrated circuit doubles every eighteen months—is that of reduced power. Smaller devices have lower parasitic capacitance, which in turn results in reduced switching currents. Microcontrollers now exceed $10^9$ instructions per second ("1 GIP") per watt, or "one MIP per milliwatt,"[2] allowing substantial computation to be powered by relatively small batteries.

Some of the more aggressive radio designs to date have yielded systems that consume approximately 4 nano Joules per transmitted bit [Carvey 1996]. With a continuous transmission at 100 KBits/second, these radios will consume 400 μWatts—a figure on par with the power consumption of modern host microcontrollers.

#### SMALL SILICON FOOTPRINT

The manufacturing cost of silicon microcontroller chips is correlated to die size. More smaller chips can be packed onto a single silicon wafer, and smaller chips have higher yields. In order to keep costs low, the circuitry that implements embedded networking should account for a small percentage of the overall chip size. This favors networking algorithms with small routing tables and computational simplicity.

#### LOW COMPUTATIONAL OVERHEAD

Computing consumes power. Networking algorithms that require less computation will be suitable for wider range of applications, especially those that are limited by available power.

2. As of this writing, several processor families meet or exceed 1000 MIPs per Watt, including Intel's XScale based StrongARM, Hitachi SDH-4, Texas Instruments MSP430 and Toshiba TX19. The list is growing rapidly.

## *Constraints imposed by the application*

The application—the "outer environment" of Embedded Networking—imposes a second set of design constraints[3].

Things That Think will become woven into our everyday environment, standing ready to serve wherever and whenever we want them and fading into the background whenever we don't. The networks linking these devices will create their own invisible mesh of communication without pre-planning, intentional placement or maintenance. If a device demands our attention, it should be due to an application-specific imperative and not due to a failing of the network[4].

The major design principals for Embedded Networking imposed by its Outer Environment can thus be summarized as follows:

#### INSTANT INFRASTRUCTURE

It is unreasonable to expect people to configure and administer a network of Things That Think. An Embedded Network must serve its users, not the other way around. This requires a network system that is created upon demand and automatically reconfigures itself as devices are added to or removed from the network.

#### SELF-CONFIGURING, "DISPOSABLE" NODES

Properly designed Things That Think will have networking built in, not added on, which will be reflected in their usage: devices will become integrated into a network simply by physically bringing then into the networking environment[5]. The

3. In this setting, *application* means "the task to which the system is applied" as opposed to "software written in support of a task."

4. In the terminology of philosopher Martin Heidegger, Embedded Networking should support devices that are "ready to hand" without causing them to become "present at hand."

network should support dynamic discovery and routing so that network services remain available as much as possible, even as devices go off-line or move.

The lifetime of a network connection will be the same as the lifetime of the object into which it is embedded. The day you dispose of an object, you dispose of the network connection without giving it a second thought.

#### CASUAL PLACEMENT

Conventional wireless networks are carefully planned with respect to location, usage patterns and density. By contrast, the quantity and density of an Embedded Network cannot generally be known beforehand. The design of an Embedded Network should support a broad range of possible device configurations, from the few to the many and from very sparse to very dense.

## *Contributions of this thesis*

#### GRAD - GRADIENT ROUTING FOR AD HOC NETWORKS

Chapter 4 describes "GRAd," a decentralized, self-organizing, multi-hop network architecture that addresses many of the design issues outlined above. GRAd's routing algorithms offer dynamic discovery and routing, and are shown to be robust even in networks with a high degree of topological change. Its multi-hop approach offers significant savings in radio transmit power. The decentralized approach used by GRAd avoids the congestion of a single base station or access point, allowing it to support up to thousands of nodes. By allowing redundancy among relaying nodes, GRAd exhibits improved reliability over unreliable links. GRAd exploits a simplified Medium Access (MAC) layer to attain lower power per transmitted bit than other comparable networking algorithms. By storing only information about

5. Some applications may call for "imprinting" a device prior to its use, for example to establish ownership. See [Stajano 1999] for an excellent description of how this can be implemented.

routing endpoints, GRAd's routing tables stay relatively small, and its networking algorithms are computationally simple—both of these points work together to make GRAd ideal for direct implementation on embedded microcontrollers.

#### DISTRIBUTED SYNCHRONIZATION

While multi-hop routing can significantly reduce the power used for radio transmissions, it doesn't address the power used for reception, which has been shown to dominate the power budget of conventional self-organizing wireless networks [Wheeler 2000]. Chapter 5, "Distributed Synchronization," shows how nodes in a wireless network can synchronize to one another without depending on a centralized time base. Once synchronized, nodes can significantly reduce their power consumption by enabling their radio receivers at selected times.

#### STATISTICAL MEDIUM ACCESS

Because the placement of nodes in an embedded network are not generally preplanned, the network can experience a wide range of node density. Chapter 6, "Statistical Medium Access," explores the effects of variable density. It will be shown that nodes can use a technique of "statistical medium access," to maximize the likelihood of successful transmission in a crowded environment. The probability of success converges as 1/e for an arbitrary number of co-located nodes.

#### ARBORNET

Chapter 7 presents "ArborNet," a prototype implementation of an embedded network. Built from commercial off-the-shelf components, ArborNet employs the basic techniques developed by this thesis to implement a self-organizing, wireless sensor network.

## *The promise, revisited*

An Embedded Network is a new paradigm in networking, and offers several benefits over conventional wireless networks.

- *Instant Infrastructure*—A node in an Embedded Network can join a network simply by bringing it within range of other nodes. This is important for creating "ad hoc" networks quickly on demand, such as in military and emergency applications. Embedded Networks are especially well suited for consumer applications, since new devices can be integrated into a network with minimal effort.
- *Proxy Intelligence*—A clock should know how to set itself. A child's toy should be able to recognize its owner's voice. No particular "intelligence" is required in the clock or toy when an Embedded Network links these devices to other computational services.
- *Data Aggregation*—Today's computers have been described as "deaf and blind" [Pentland 1998], sensorially deprived and unable to act sensibly. Embedded Networks can be used to gather crude data from hundreds or thousands of sources for distillation into high-quality information.
- *Cheap Links*—In many cases, the cost of physical links is a significant part of a total system budget. For example, a light switch costs only about $2 for the switch itself, but the cost of conduit, copper wire, and installation time brings the installed cost to over $70. In many cases, Embedded Network links offer inexpensive alternatives to wired connections.

## *What will happen?*

What will happen when every embedded microcontroller comes equipped with its own wireless, self-organizing, scalable network connection? Answering this question is a bit like trying to anticipate the effects of the Internet a decade ago. It was widely believed that the Internet could make a large difference in the way we communicate, but few people could anticipate the depth and breadth of its effects.

So it is with embedded networking. While it may not be possible or practical to anticipate the specific manifestations of embedded networking, it is entirely reasonable to believe that the implications will be large. Some of the applications are easy to imagine:

- *ArborNet—understanding the biosphere of the forest floor.* A paper company owns thousands of acres of forest but, short of sending in survey crews, has little knowledge of the ecology and health of the forest. So they create a small "dart" with a tiny analysis lab in the tip and a radio link in the tail. Thousands of these darts are scattered from an airplane over the forest, forming a complete communication mesh that informs the company about drought, flood, or fire conditions.
- *OmniSense—an office building on-line*. Once every light switch, thermostat, door jamb and motion detector of a building are connected to a network, power and security systems can be precisely managed. Over time, the system can learn the patterns of usage, allowing it to anticipate ordinary events and to flag abnormal conditions.
- *Vox Populi—an inter-village telephone system.* Imagine a telephone system that is as easy to set up as handing out telephone handsets. There is no expensive base station–the telephones themselves become the network. And as a boon (or a bane) to the prevailing government, a system can be designed for which centralized control is neither necessary nor possible.
- *GridKey—a solution to urban gridlock.* Each street corner of a city has a simple sensor that detects the passage of cars. All of the sensors are networked together so analysts—both human and computer—can form a city-wide picture of traffic patterns, adjusting traffic signal timing and issuing advisories to reduce congestion. Individuals can access this information via mobile devices and plan their routes accordingly.

Perhaps the best way to learn about the implications of embedded networking is to build them. Herein lies the crux of this thesis.

# CHAPTER 2 *Precedents in Wireless Networks*

*Existing standards do not address the needs of Embedded Networks. In their quest to communicate further and faster, none yield networks that are simultaneously self-organizing, low-power and scalable.*

In the early 1970s, the Packet Radio Program, funded by the Advanced Research Projects Administration (ARPA), and Norm Abramson's AlohaNet laid the groundwork for wireless digital networks [Kleinrock 1987][Abramson 1985]. Since that time, wireless digital communication systems have grown both in range and capability. Satellite-based systems provide global wireless networks. Locally, high-speed wireless links are commonplace in today's office buildings.

Wireless Local Area Networks (WLANs) offer high speed communication over short distances. The popular 802.11 standard offers communication rates of 11 megabits per second over a range of 200 meters [IEEE 1999]. Wireless Wide Area Networks (WWANs) offer longer range at reduced bit rates, as exemplified by the UMTS standard with a bit rate of two megabits per second carried over cellular telephone networks [UMTS 2000].

### BLESSED ARE THE MEEK...

Wireless LAN's are optimized for speed, wireless WAN's are optimized for distance. By contrast, the important attributes for embedded networks are neither speed nor distance, they are power and scalability at a low cost. When voice or image data need to be transmitted, current networks may be the most appropriate. However, for many everyday objects, communication rates on the order of bits per hour—not megabits per second, will suffice. Figure 2 highlights the natural home

of embedded networks: low bit-rate short-haul communications, an area left untouched by conventional wireless networks.

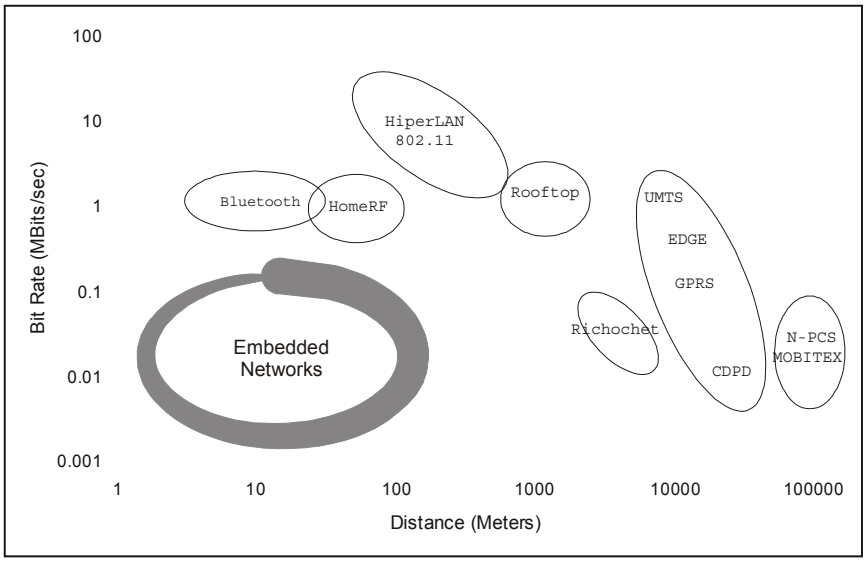



**FIGURE 2. Distance versus bit rate for wireless standards**

As established in Chapter I, a viable embedded network will have a multi-hop architecture with decentralized control. It will have dynamic routing and will also incorporate power conservation techniques in its core design. As shown in Table 1 below, in their quest to communicate further and faster, none of the existing standards simultaneously address all four of these attributes.

**TABLE 1. Packet switched wireless networks**

| wireless standard | attributes | | | | |
|---|---|---|---|---|---|
| | Decentralized | Dynamic Routing | Multi-hop | Managed power | comments |
| Historical Systems | | | | | |
| AlohaNet | x | x | | | full flood |
| Packet Radio | x | x | x | | long-haul |
| Wireless Local Area Networks | | | | | |
| Bluetooth | | x | | x | Eight nodes per piconet |
| 802.11 | | x | | x | base station mode |
| 802.11 peer-to-peer | x | x | | | peer to peer mode |
| Hiperlan/1 | | x | | x | similar to 802.11 |
| Hiperlan/2 | | x | | x | proposed multi-hop option |
| DECT | | x | | x | designed for packetized voice |
| HomeRF | | x | | x | Hybrid of 802.11 and DECT |
| Wireless Wide Area Networks | | | | | |
| Metricom Ricochet | | | x | | fixed “pole top” units |
| Nokia Rooftop | x | x | x | | One Access Point per dozen nodes. |
| MANET working group | x | x | x | | Not a commercial standard (yet) |
| CDPD, UMTS, EDGE, GPRS | | x | | x | Cellular Telephony networks |
| N-PCS, MOBITEX | | x | | x | Two-way pager networks |

## *Legacy systems*

### ALOHANET

AlohaNet was developed in the 1970s by Norman Abramson and his colleagues, and is one of the earliest packet-switched wireless digital networks. It used ground-based radios transmitting on a single shared channel. While this architecture is not generally scalable—congestion increases with the number of nodes—AlohaNet and

its analysis spawned many other systems, including Ethernet and TDMA protocols for satellite communication.

#### PACKET RADIO

Packet Radio systems are among the earlier examples of multi-hop wireless communication systems[1]. In 1972, the ARPA launched the Packet Radio Program, designed to develop robust communication systems for the battlefield. In the late 1970s, amateur radio operators developed "Terminal Node Controllers" (TNCs) to form digital links among meshes of ham radios. Since then, TNCs have evolved to support several forms of multi-hop communications, including static and dynamically discovered routing (ROSE and NET/ROM respectively).

## *Local Area Networks*

#### BLUETOOTH

Developed by an industry consortium, Bluetooth specifies a radio and access protocol. The radios are spread-spectrum in the 2.4GHz band, and will form ad-hoc "piconets" of up to eight devices. Within each piconet, one device is the local master and chooses a spreading code. Other devices within that piconet use the master for control and synchronization. One master may participate in multiple picnonets to form a "scatternet," but the specification does not support multi-hop communication.

#### 802.11 WIRELESS LAN

IEEE 802.11 has been widely adopted as an industry standard for Wireless Local Area Networks. Links are specified as 2.4 GHz spread-spectrum transceivers using

1. To be fair, Packet Radio was hardly the first multi-hop wireless communication network: Napoleon's Optical Telegraph, built before the turn of the 19th century, predated packet radio by 170 years.

CSMA protocols. Channel data rate is as high as 11 MBits/sec. 802.11 works well for linking several dozen devices to a wired Access Point (base station), but will not scale well to higher densities.

802.11 also specifies an ad hoc mode, which provides point to point links at the expense of frame relay and power savings support.

### HIPERLAN

Hiperlan/1 (High Performance European Radio Local Area Network) has been developed by ETSI (the European Telecommunications Standards Institute) as a second generation wireless local area network. It supports bit rates of 20 MBits/second at distances of up to 50 meters. The standard specifies the physical layer (PHY) and the Medium Access Layer (MAC), and while it admits the possibility of a multi-hop architecture, it does not specify how it should be implemented.

Hiperlan/2 is a new WLAN standard being developed at ETSI. It specifies channel bit rate of 54 MBit/second with intra-nodes distances up to 100 meters.

### DECT

Development of the DECT (Digital Enhanced Cordless Telecommunications) specifications was started in the mid-1980s and finished in 1992 by ETSI. Originally developed as a standard for cordless telephones, the scope of DECT has been expanded to support general digital radio access. The current standard offers a base station architecture with wireless data links of 1.152MBits/second over a range of 100 meters. According to Ericsson, DECT permits the highest user densities of any cellular system, up to 100,000 nodes per square kilometer[2].

2. See the online document http://www.ericsson.com/BN/dect2.html for more information.

#### HOMERF

The HomeRF Working Group is creating specifications for low-cost intra-home networking named SWAP (Shared Wireless Access Protocol). SWAP has adopted a hybrid approach, using 802.11 protocols to carry data and DECT protocols to carry voice. A SWAP network will support up to six voice conversations and up to 127 devices in each network.

As in 802.11 networks, a SWAP network can work in ad hoc mode, in which all devices have equal access to the network, and in managed mode, in which one central device coordinates the operations of the other nodes in the network.

## *Wide Area Networks*

#### RICOCHET

Developed by Metricom Corporation of Los Gatos, CA, the Ricochet Network is one of the first commercial multi-hop wireless digital networks. A mesh of Network Radios, typically mounted on utility poles one to four kilometers apart, relay packets between Wireless Modems and wired Access Points. The first generation of Wireless Modems offered users a channel data rate of 28.8 kilobits per second; newer Modems provide 128 kilobits per second.

A Ricochet network is a multi-hop system, but not self-organizing: adding a new Network Radio to the mesh requires manually incorporating it into the network and setting up static routing to the nearest wired Access Point.

#### ROOFTOP

Rooftop Communications (recently purchased by Nokia) offers wireless networking products that form a multi-hop, dynamically routed mesh of terrestrial radios. Each radio runs in the 2.5 GHz ISM band, supports link rates of 1.6 MBits/second, and has a range of approximately three miles.

#### CDPD, UMTS, EDGE, GPRS

CDPD (Cellular Digital Packet Data), UMTS (Universal Mobile Telecommunication Systems), EDGE (Enhanced Data Rates for Global Evolution) and GPRS (General Packet Radio Service) are wireless systems that use cellular telephone networks to carry digital data. Data rates range from 19.2 kilobits per second (CDPD) to a predicted rate of 2 megabits per second (UMTS).

#### N-PCS, MOBITEX, ARDIS

N-PCS (Narrowband Personal Communication Services), MOBITEX and ARDIS (Advanced Radio Data Information Services) are essentially two-way pager systems. Low bit-rate data is transferred between individual mobile units and high power base stations. Data packets are usually of limited size, and data rates range between 8 and 24 kilobytes per second.

### *Other multi-hop protocols*

#### MANET

The mobile ad-hoc network (MANET) working group is an effort within the IETF (Internet Engineering Task Force) to develop and evolve routing specifications for wireless ad-hoc networks containing “up to hundreds” of nodes. The working group has already published ten Internet Drafts for discussion and debate, and covers such topics as adaptive routing and quality of service.

A standard benchmark for MANET network protocols assumes that they are implemented using 802.11 wireless links running in point-to-point “ad hoc” mode. In this mode, individual neighbors must be known in order to achieve media access, and the three way handshake at each packet transfer increases latency. Power conservation is not possible as the receiver cannot be turned off.

#### OTHER PROTOCOLS

The last few years have seen many developments in ad hoc, multi-hop routing protocols. Active areas of research include data-directed routing for network efficiency, data aggregation to reduce network traffic and choosing cluster heads dynamically to reduce per-node power requirements [Intanagonwiwat 2000], [Heinzelman 2000]. These techniques show promise as important components of Embedded Networking systems.

## *What's missing?*

Although existing wireless standards address a range of applications from low bit rate, long distance communication to high-bandwidth, short haul systems, none of them have the right mix of scalability, self-organization and low-power required as a basis for embedded networking. A re-thinking of the network is needed.

# CHAPTER 3 *Multi-hop Communications*

*Imagine you are at a party where the conversation flows as freely as the champagne. Suddenly, a guest picks up a bullhorn and shouts out in a booming voice to his friend on the opposite side of the room, asking for some more duck canape. The sound is deafening, and all other conversation comes to an abrupt stop.*

## *The virtues of whispering*

Many familiar wireless communication systems, including cellular telephones, two-way pagers and wireless LANs use a *single-hop* design: a central base station or access point maintains direct radio communication with each terminal node of the network. A single-hop system can be likened to that bullhorn: whenever the base station transmits, it precludes other communication within its area.

By contrast, in a *multi-hop* wireless network, each node transmits with reduced power, communicating with a set of neighboring nodes within a limited range. Those neighboring nodes in turn relay the message on behalf of the originator, and so on, until the message arrives at intended destination.

*Multi-hop communication conserves transmitter power and increases system bandwidth.*

Multi-hop networks offer advantages over their single hop counterparts. By reducing the transmit range in each node, multi-hop networks offer substantial power savings. Multi-hop networks exploit spatial reuse, yielding higher effective bandwidth. And by reducing the overall levels of radio interference and noise, multi-hop networks can scale to handle more nodes than single-hop networks.

## *Single-hop and multi-hop: an idealized comparison*

In a single-hop network, each radio transmits with sufficient power to reach its ultimate receiver without intervening relays. In a multi-hop system, each radio transmits with enough power to reach one or more neighboring nodes, which will in turn relay the message until it reaches its final destination.

A representation of single hop communication is shown in Figure 3.

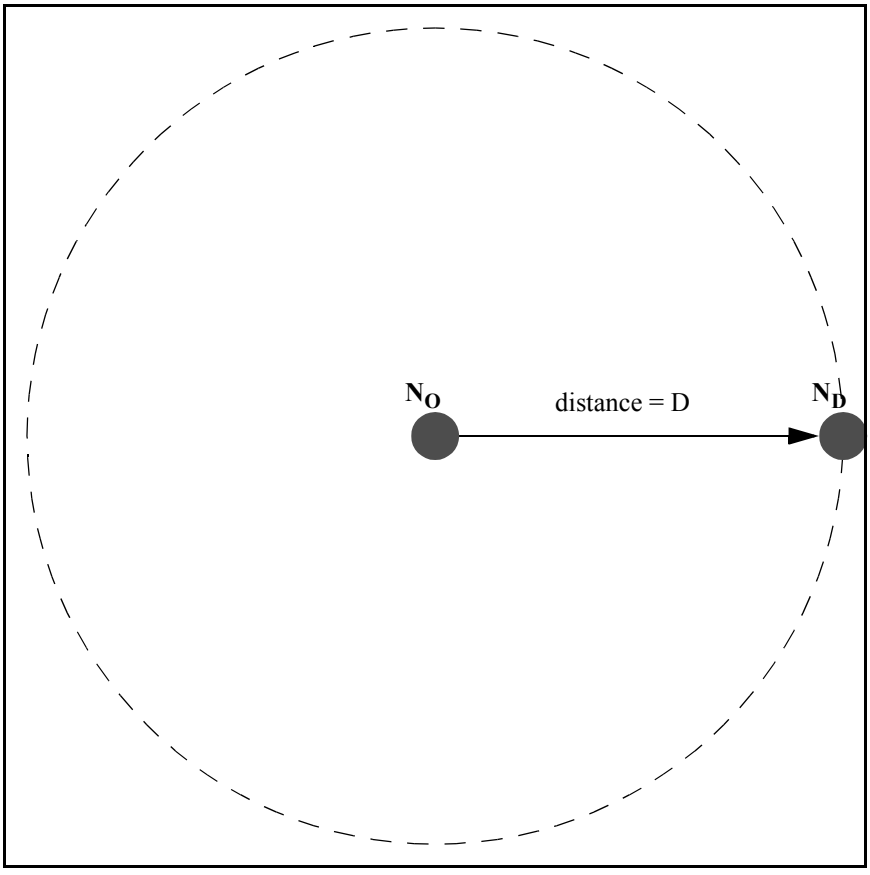



**FIGURE 3. Single hop communications**

In Figure 3, $N_O$ is the originating node, $N_D$ is the destination node, and the two nodes are separated by a distance D. The transmit power required to span a distance of D is defined to be P.

A multi-hop uses multiple relay stations to get the message from $N_O$ to $N_D$, as illustrated in Figure 4.

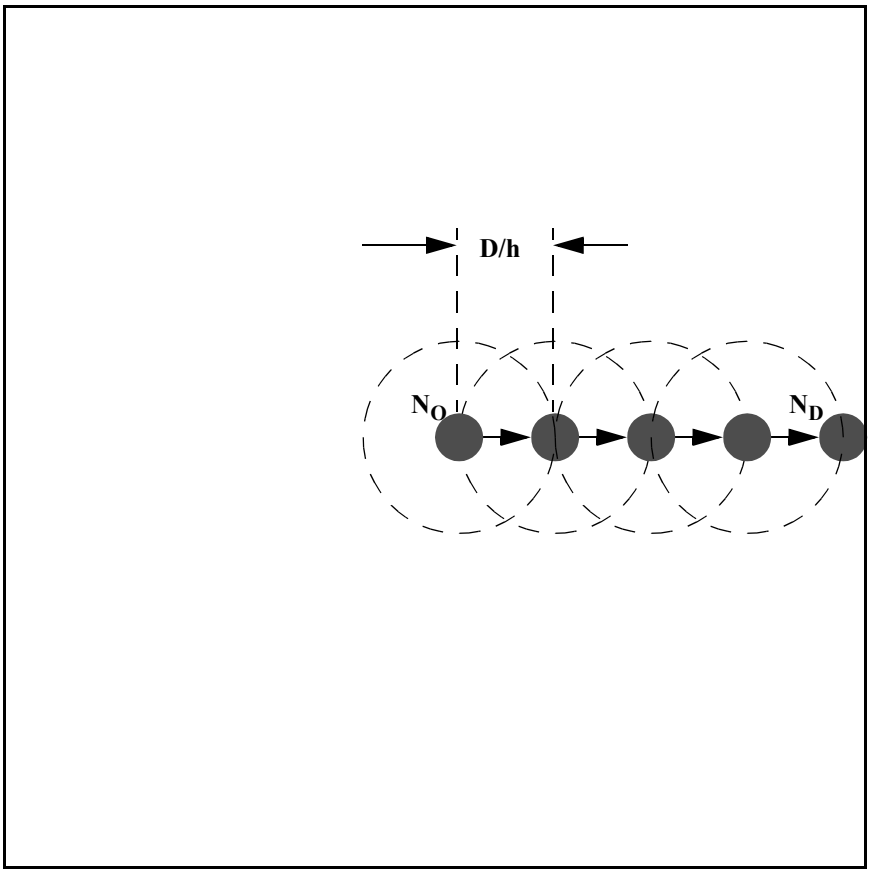



**FIGURE 4. Multi-hop communications**

Given a system that requires P units of power to transmit its message in a single hop and a path loss exponent of *e*, the per-node power required to send the message using *h* hops can be approximated by

$$P_n(h) = Ph^{-e} \quad \textbf{(EQ 1)}$$

The system-wide transmit power is the sum over *h* hops, or

$$P_t(h) = Ph^{1-e} \quad \textbf{(EQ 2)}$$

The path loss exponent in free space has a theoretical value of 2 for free space, but is typically cited as 4 or higher for office or urban environments.

Assume for the moment that each transmitter covers a perfectly circular area, and the distance covered by a single hop system is D. Assume that the transmitters in an *h* hop system are equally spaced at distance $d = D/h$. If each transmitter uses the minimum amount of power to reach the next receiver, the total area covered by the transmitters is given by

$$A_t(h) = \pi d^2 + (h-1)d^2\left(\frac{\pi}{2} - 1\right). \quad \textbf{(EQ 3)}$$

If we define $k = \left(\frac{\pi}{2} - 1\right)$, Equation 3 can be written as:

$$A_t(h) = d^2hk + d^2(\pi - k). \quad \textbf{(EQ 4)}$$

Substituting D/*h* for *d* in equation *4* yields:

$$A_t(h) = D^2\left(\frac{k}{h} + \frac{(\pi - k)}{h^2}\right). \quad \textbf{(EQ 5)}$$

Equation 5 tells us that the area covered by transmitters in a multi-hop system decreases roughly linearly with the number of hops.

## *Power savings*

The rather idealized system using *h* hops has several advantages compared to its single-hop counterpart. The power required by each node is reduced by $h^e$, while

the total power consumed by the system is reduced by a factor of $h^{(e-1)}$ and the total area covered by transmission is reduced by a factor slightly larger than $h$.

As an example, assume a single hop system with a transmit distance of 100 meters in an office environment with a path loss exponent of 4. If we replace the single hop system with a 10 hop system, the transmit power per node is reduced by a factor of 10,000, the total system power is reduced by a factor of 1,000, and the area covered by the transmissions is reduced by approximately a factor of 12.

Figure 5 summarizes the relative transmit power for a variety of hops and path loss exponents.

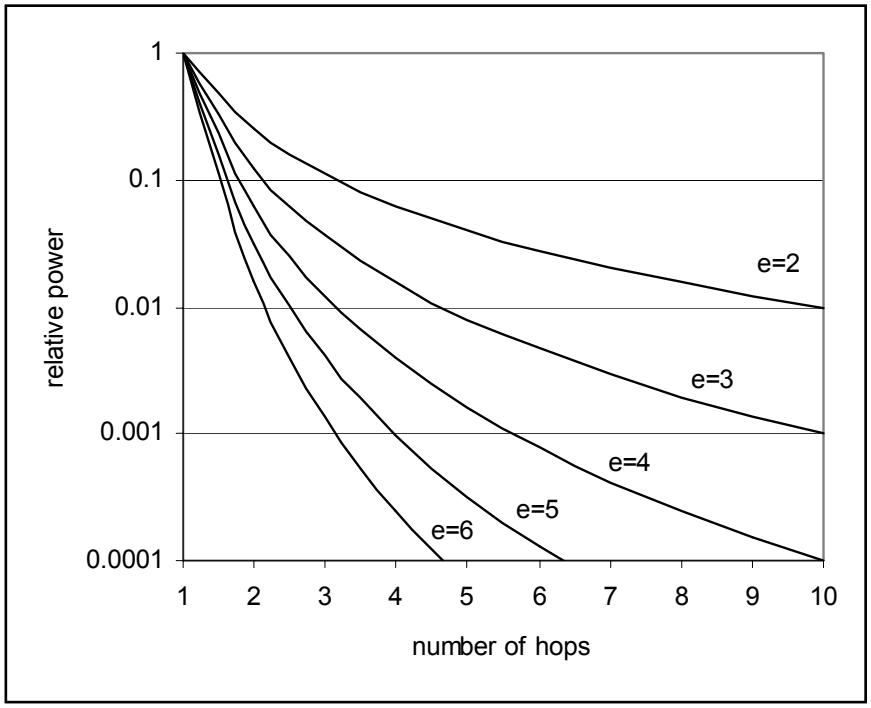



**FIGURE 5. Per-node transmitter power (relative to single hop)**

The power savings in a multi-hop network can be substantial. For example, in an environment with a path loss exponent of 4, transmitters in a five hop system require 0.0016 of the power compared to a single hop system.

## *Effects of non-uniform spacing*

In the multi-hop scenario given above, relaying nodes are assumed to be evenly spaced between the source and the destination with the transmitter power set to the absolute minimum for reliable communication. In a practical embedded network, nodes will unevenly spaced, and some amount of redundancy and overlap must be expected if there is to be a continuous, reachable path between the originator and destination nodes.

The effect of overlap does not change the per-node transmitter power required, but it does increase the number of nodes involved in relaying the message and thus the total system-wide transmit power. Assuming a factor of N redundancy, the system-wide power of Equation 2 becomes:

$$P_t(h) = PNh^{1-e} \quad \textbf{(EQ 6)}$$

Revisiting the example of a ten hop network with a path loss exponent of 4: if this network has a factor of five redundancy, this represents a factor of 200 reduction in total transmitted system power compared to its single-hop counterpart.

The effect of overlap is to increase the total system-wide power by a linear multiplier, while the savings in power through reduced distance are exponential. The net effect is that a multi-hop system conserves transmit power compared to a single-hop system, even when taking non-idealized spacing of nodes into account.

## *Summary*

Multi-hop systems offer several advantages over single-hop systems.

### REDUCED TRANSMITTER POWER

In a multi-hop system, the reduced distances between nodes allows the transmitter power to be reduced exponentially. For example, assuming a path loss exponent of 4, if the 100 meter range of an 802.11 wireless LAN node is reduced to ten meters, its transmitter power may be reduced by 40 db, or a factor of 10,000. This reduction in power results in longer battery life for individual nodes, and reduces the overall amount of clutter in the airwaves.

### SPATIAL REUSE

In a single-hop system, transmissions from a base station to a single mobile node blanket the airwaves surrounding the base station. In a multi-hop system, the area covered by transmissions are localized by approximately a factor of h, where h is the number of hops. This permits simultaneous transmissions to take place in physically separate parts of the network—a technique sometimes referred to as *Spatial Division Multiple Access* (SDMA). Since the airwaves can support multiple transmissions, the effective bandwidth of the overall system increases.

### MAPPING THE TOPOLOGY

In a single-hop system, a node is either within range of the base station or not: nothing is learned about the topology of the network. In a multi-hop system, the topology of the network can be acquired as the nodes converse with one another. This information can be used to establish optimal routes or physically locate a node.[1]

1. Using a wireless network topology to model physical topography doesn't always work as well as one would hope, as will be shown in Chapter 7.

# CHAPTER 4 *GRAd: Gradient Routing for Ad Hoc Networks*

This chapter presents *Gradient Routing* (GRAd), a novel approach to routing and control in wireless ad hoc networks. A GRAd network attains scalability through a *multi-hop* architecture: nodes that are not within range of one another can communicate by relaying messages through intermediate neighbors. Routing information is established on-demand and is updated opportunistically as messages are passed among nodes.

Unlike other ad hoc routing techniques, a node in a GRAd network does not single out a particular neighboring node to relay its message. Instead, it advertises its "cost" for delivering a message to a destination, and only those neighboring nodes that can deliver the message at a lower cost will participate in relaying the message. In this way, a message descends a loop-free "gradient" from originator to destination.

Since multiple neighbors can participate in the relaying of messages, GRAd maintains good connectivity in the face of frequently changing network topologies. A node does not need to know the identities of its neighbors and establishes routes on demand, making periodic "hello" beacons unnecessary and increasing the overall security of the network. Because GRAd does not use link to link handshakes, end-to-end latencies remain small.

## *The challenge*

In any wireless ad hoc network, a major challenge lies in the design of routing and network control. Lacking any centralized point of control, nodes in an ad hoc net-

work must cooperatively manage routing and medium access functions. Nodes may be mobile, creating continual changes in the network topology. Also, wireless links are not as robust as their wired counterparts; high bit error rates and packet losses are commonplace.

In the last decade, a number of ad hoc network protocols have been proposed. As an indicator of the amount of activity in this field, the Internet Engineering Task Force (IETF) recently formed the Mobile Ad Hoc Networking (MANET) working group to develop ad hoc protocol specifications and introduce them into the Internet Standards track [Macker 2000]. At this time, there are eight separate ad hoc routing protocols under consideration by the working group.

GRAd falls under the category of *on-demand* routing protocols, in which routes are established only when nodes wish to communicate with one another; no attempt is made to maintain state when there is no data to send.

In other on-demand routing protocols such as the *Ad Hoc On-Demand Distance Vector Routing* protocol (AODV) [Perkins 1999] and the *Dynamic Source Routing* protocol (DSR) [Johnson 1999], a node relays a message by sending to a particular neighboring node. The popular 802.11 MAC layer protocol uses "virtual carrier sensing" as part of its collision avoidance mechanism for such unicast transmissions [IEEE 1999], requiring a *request to send / clear to send* handshake (RTS/CTS) between each pair of wireless links. This exchange contributes to significant delays in the relaying of messages, resulting in long latencies.

By comparison, a node in a GRAd network makes no attempt to identify *which* of its neighbors is to relay a packet. Instead, it includes its "cost to destination" information in the packet and broadcasts it. Of all the nodes that receive the broadcast, only those that can deliver the packet at a lower cost will relay the message. In this way, the packet descends a loop-free "gradient" towards the ultimate destination.

Since each transmission is a local broadcast, GRAd does not (and in fact, cannot) use the RTS/CTS handshake associated with unicast transmissions. Consequently, GRAd exhibits very low latencies.

GRAd collects cost information opportunistically: each message carries with it the cost since origination, which is recorded at each node that overhears the transmission, and is incremented when the message is relayed. Thus, the simple act of passing a message quickly and efficiently updates the cost estimates in nearby nodes.

GRAd demonstrates very good immunity to rapidly changing topologies. Since each message reaches a number of neighboring nodes, a single link failure will not cause a break in the communication path as long as another neighbor is available to relay the message.

## *The GRAd algorithm*

#### ASSUMPTIONS

GRAd is designed for use in multi-hop wireless networks, and makes relatively few assumptions about the underlying physical medium. It does assume that links are *symmetrical*: if Node A can receive messages from Node B, then Node B can receive messages from Node A. In a practical wireless network, strict symmetry is impossible to guarantee due to the mobility of the nodes and time-varying environmental noise. As will be shown in by simulation, GRAd continues to work well in cases where only partial symmetry holds.

GRAd assumes a local broadcast model of connectivity. When a node transmits a message, all neighboring nodes within range simultaneously receive the message.

GRAd provides best effort delivery of messages with the understanding that higher-level protocols will handle retransmission and reordering of packets as needed.

The propagation of a message through the network establishes and updates *reverse path* routing information to the originator of the message. Consequently, GRAd is most efficient when the network traffic has a "call and response" pattern, such as streamed packet data with periodic acknowledgments.

### GRAD MESSAGE FORMAT

Messages passed among nodes in a GRAd network carry a header containing the fields shown in Table 2. A description of each field follows.

| msg_type | originator_id | seq_# | target_id | accrued_cost | remaining_value |
|---|---|---|---|---|---|

**TABLE 2. GRAd message format**

`msg_type`: Takes on one of two values, `M_REQUEST` for a reply request message and `M_DATA` for all others.

`originator_id`: The id of the node originating this message. This id may be statically assigned, or may be dynamically generated on a per-session basis.

`seq_#`: A sequence number associated with the originator id, and incremented each time the originator issues a new message. The combination of `[originator_id, sequence_#]` uniquely identifies a message, so a receiving node can distinguish a new message from a copy of a message already received.

`target_id`: The id of the ultimate target for this message.

`accrued_cost`: Upon origination, the `accrued_cost` of a message is set to 0.0. When the message is relayed, the relaying node increments this field by one. Thus, `accrued_cost` represents the estimated number of hops required to return a message to `originator_id`.

`remaining_value`: Upon origination, this field is initialized to the estimated number of hops to `target_id`. Whenever the message is relayed, this field is dec-

remented by one. The `remaining_value` field represents the "time to live" of the message: if it ever reaches zero, the message is dropped.

### COST TABLE

Each node maintains a *cost table*, analogous to the routing table of other algorithms[1]. The cost table plays two important roles in GRAd. First, the cost table can answer the question *"Is this message a copy of a previously received message?*" This is determined by comparing the `seq_#` in the message from a particular originating node against the last `seq_#` recorded in the cost table for that originator. Second, it can answer the question *"What is the estimated cost of sending a message to target node X?"* This cost estimate is formed by recording the `accrued_cost` fields for each `origninator_id` in received messages.

### COST TABLE FORMAT

Each entry in the table holds state information about a remote node, as shown in Table 3.

| target_id | seq_# | est_cost | expiration |
|---|---|---|---|

**TABLE 3. Cost table entry**

`target_id`: The id of a remote node to which this cost entry refers.

`seq_#`: The highest sequence # received so far in a message from `target_id`. When compared against the `seq_#` of a newly arrived message, this field discriminates between a new message and a copy of a previously received message.

`est_cost`: The most recent and best estimated cost (number of hops) for delivering a message to `target_id`.

1. The term "cost table" is chosen over the more conventional "routing table" to emphasize that GRAd does not prescribe a specific route to a target node, but rather it maintains an estimated cost to the target.

`expiration`: When a cost entry is updated, this field is set to the current time plus `cost_entry_timeout`. If the current time ever exceeds `expiration`, the cost entry is purged from the table.

#### COST TABLE MAINTENANCE

When a message is received at a node, the `originator_id` of the message is compared against the `target_id` of each entry in the cost table.

If no matching entry is found, a new cost entry is created, for which `target_id` is copied from the message's `originator_id`, `seq_#` is copied from the `seq_#` field, and `est_cost` is copied from the `accrued_cost` field. The message is marked as "fresh."

If a `target_id` is found that matches the `originator_id` of the incoming message, and if `seq_#` in that entry is lower than the `seq_#` of the incoming message, the message is marked fresh and the cost entry fields are updated from the corresponding fields in the message.

Otherwise, the message is marked as "stale"—it is a copy of a message previously received. However, if the messages offers a lower cost estimate in its `accrued_cost` field than the recorded cost in the `est_cost` field, the lower cost is recorded. This has the effect that if a copy of a previously received message subsequently arrives by means of a shorter path, the shorter path is recorded.

#### MESSAGE ORIGINATION AND RELAYING

When a node wishes to send a message to a destination for which the cost to the target is known, it transmits a message with the `msg_type` field set to `M_DATA`, specifying the destination in the `target_id` field and the cost to that destination in the `remaining_value` field.

Of the neighboring nodes that receive the message, only those that can relay the message at a lower cost, as indicated by their cost tables, will do so. Before a neigh-

boring node relays a message, it debits the `remaining_value` field by one. As this process repeats, the message "rolls downhill," following an ever decreasing gradient from the originator to the target.

At the same time, the message carries the originator of the message in the `origination_id` field and the accumulated relay cost since origination in the `accrued_cost` field. Upon origination, the `accrued_cost` is set to 0. Each node that receives the message increments the `accrued_cost` field of the message and then updates its cost table entry for the originating node based on this information. If and when the message is relayed, it is re-sent using the incremented `accrued_cost`. By this process, any node that receives a message can update its cost estimate for returning a message to the originating node, whether or not the node is actively involved in relaying the message.

#### REPLY REQUEST MESSAGES

When a node wishes to send a message to another node for which there is no entry in the cost table, it initiates a "reply request" process. To do so, the originating node transmits a message whose `msg_type` field is set to `M_REQUEST`, specifying the destination in the `target_id` field and initializing the `remaining_value` field to `default_request_cost`.

Relaying of the message proceeds much in the same manner as for a `M_DATA` message, but with one important exception: any node that receives an `M_REQUEST` message will *always* relay the message the first copy of the message it receives, unless the `remaining_value` field has reached zero. As with an `M_DATA` message, the node will increment the `accrued_cost` and decrement the `remaining_value` fields of the message before relaying the message.

If a node receives a copy of a previously received message, it will update its cost table entry for the originator of the message if the copy represents a lower cost to the originator, but the node will not relay the copy.

Two important things happen in the reply request process. First, if the destination node is present anywhere in the network (within a radius of `default_request_cost` hops), it will receive the `M_REQUEST` message and initiate a reply. Second, each node that receives the `M_REQUEST` message establishes a cost estimate for returning a message to the originator. Consequently, when the destination node responds to the originating node's request, it can use the more efficient `M_DATA` message to deliver the reply.

#### CALL AND RESPONSE

GRAd is uses *on demand* routing: none of the nodes have any *a priori* knowledge of one another. Until a node turns on its transmitter, its presence in the network is not known to other nodes. The general rule for such networks is "if you wish to be spoken to, you must first speak."

The following two figures illustrate the Reply Request process in an ad hoc network, in which Node A initiates a request to Node B, and Node B subsequently responds. It is assumed that initially none of the nodes in the network have any knowledge about nodes A or B.

Figure 6 shows the state of the network after the propagation of a Reply Request message from Node A to Node B. The dashed circle around Node A shows the range of an individual transmitter. Node A starts by transmitting a Reply Request message with an accrued cost of 0 and a target_id set to the ID of Node B. The two neighbors to A each increment the accrued_cost field of the message, record the fact that they are each one hop away from A, and relay the message. This process

continues until all the nodes in the network have received and relayed the Reply Request message.

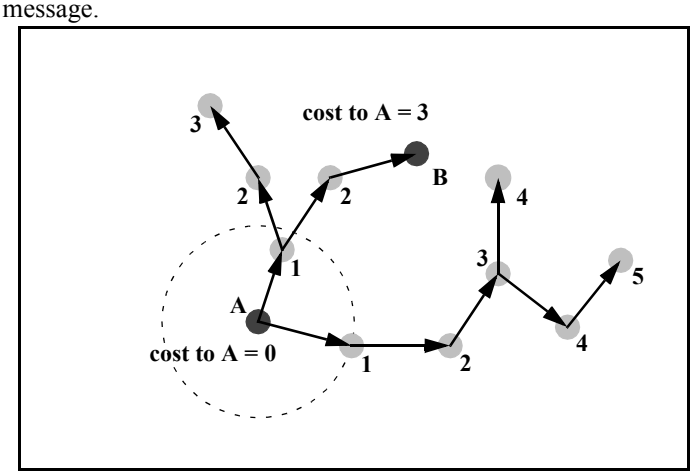



**FIGURE 6. Reply Request from node A to node B**

By the time A's Reply Request message has arrived at node B, all of the intervening nodes in the network have established a cost estimate for returning a message to A, as shown by the numbers next to each node in Fig. 1.

When Node B receives the Reply Request message from Node A, it responds by originating an "ordinary" message with `msg_type` set to `M_DATA`, accrued_cost set to 0, and `remaining_cost` set to 3, the known cost required to reply to Node A.

Referring to Figure 7, when B transmits this message, the neighboring nodes C and D lie within range and receive the transmission. The cost table of C indicates that A is two hops away, and since the message has an advertised `remaining_cost` of three, node C should relay the message after decrementing its `remaining_cost`. Node D, on the other hand, is four hops away from node A, and since it is unable to relay the message at a cost lower than the `remaining_cost` advertised by the message, it drops the message.

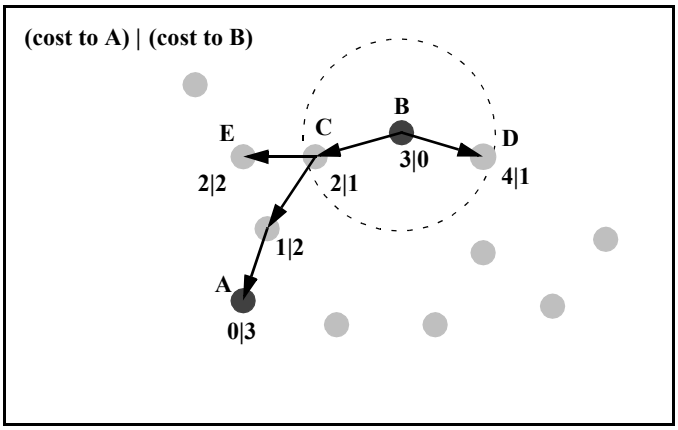



**FIGURE 7. Node B replies using the reverse path**

As the message is relayed towards A, intermediate nodes also create entries for returning a message to node B. Figure 7 shows the state of the cost tables after the reply has been received at Node A. Next to each node that participated in the reply, the estimated cost for sending a message to node A is shown to the left of the vertical bar, the cost to node B is shown to the right.

In this example, nodes D and E received the message from B, but did not actively participate in relaying it. Nonetheless, by virtue of "overhearing" the message, these nodes have established an estimated cost for sending a message to Node B should the need ever arise.

### ROUTE REPAIR

As nodes in the network enter or leave the network, or move relative to one another, the topology of the network can change dynamically, rendering the individual nodes' cost estimates inaccurate.

If the path between originator and target becomes shorter, GRAd will automatically compensate for the change by "skipping over" one or more intervening nodes, and

the revised cost estimates will be reflected in the participating nodes' cost tables after a single call and response pair of messages.

In the event that the path become longer, or intervening nodes change their positions, there is the possibility that the originator's cost estimate no longer has sufficient "potential" to reach the target destination.

GRAd uses end-to-end acknowledgments. If an acknowledgment is not received within a fixed amount of time, the originator can re-send the message, but this time using a higher estimated cost to the destination in the `remaining_cost` field of the message. This has the effect that more intermediate nodes will participate in relaying the message towards its destination. As before, by the time the message reaches its destination, all of the intermediate nodes will have fresh cost estimates for returning a message to the originator, so the destination node's acknowledgement will be able to follow an updated gradient back to the originator.

If the first attempt to re-send a message with an increased estimated cost fails to reach the destination, the originator can repeat the process, incrementing the initial estimated cost each time.

If, after several attempts, the message fails to reach the destination, the originator can issue a new Reply Request message to create fresh cost estimates from scratch.

#### IMPLICIT ACKNOWLEDGMENT

To reduce the number of redundant messages transmitted, GRAd uses a variant of passive acknowledgment called *implicit acknowledgment*. A message to be relayed is stored in a MAC-level buffer while it awaits transmission. If a node overhears a neighbor relay a copy of that same message, but at a lower `remaining_cost`, then the node can assume that the neighbor has succeeded in delivering the message closer to the destination than this node, therefore it can delete the message from the MAC queue and cancel its transmission.

When the ultimate target node receives a message, it re-transmits the message with a `remaining_cost` set to zero. This has the effect of notifying any neighbors still waiting to relay the message that the target has received the message and that they may abandon their efforts.

## *Simulation and results of GRAd*

Performance of GRAd was simulated using *Jasper* [Poor 2001], an event driven network simulator. The main objectives of the simulation were to characterize the performance of GRAd as a function of transmitted packets and the amount of mobility among the nodes in the network.

### SIMULATION ENVIRONMENT

Jasper provides detailed models for components of a multi-hop, mobile, wireless ad hoc network.

The radio modelled by Jasper emulates an FM or spread-spectrum radio, such as would be used in a wireless local area network, operating in an urban or dense office environment. In the absence of other transmissions, a transmitter/receiver pair has a nominal range of 250 meters. Transmit power falls off as the cube of the distance, and a receiver can acquire lock on a transmitter if the signal to interference ratio exceeds 10db. Once locked, a receiver can hold lock as long as the signal to interference ratio exceeds 6db. During reception of a packet, if the signal to interference ratio drops below 6db, the packet is marked as corrupted. The bit rate of the transmitter is 2Mb/sec.

GRAd's MAC layer uses a technique of carrier sense with exponential backoff: when a node wishes to transmit a packet, it first waits for a random interval between $T_b$ and $2T_b$ seconds. At the end of that time, if the carrier sense detects that the local airwaves are in use, it doubles the value of $T_b$ (up to an upper limit) and waits again. If the airwaves are free, it halves the value of $T_b$ (down to a lower

limit) and transmits the packet. If the MAC transmit buffer becomes empty, $T_b$ is reset to its minimum value.

Mobility and traffic models were chosen to emulate those described in [Brooch 1998]. Fifty nodes in a 1500m × 300m arena travel according to the *random way-point* algorithm: each node travels towards randomly chosen locations within the arena at random speeds (evenly distributed between 0 and 20m/sec.). After reaching its destination, the node pauses for a fixed amount of time before setting out for its next randomly chosen location. The *pause time* is varied from 0 seconds for continuous motion to 900 seconds in which case the nodes are stationary for the duration of the simulation.

Traffic is generated by *constant bit rate* (CBR) sources, randomly chosen among the 50 nodes. Each CBR source targets a randomly chosen destination among the remaining 49 nodes. A CBR source generates four 64 byte packets per second. The load on the network is controlled by the changing number of CBR sources. In the tests, 10, 20, 30 and 40 CBR sources were used to generate traffic.

Messages from the CBR source sit in a queue until a route is discovered. To prevent indefinite buffering, messages are dropped if they remain in the queue for over 30 seconds.

Entries in cost tables are set to time out if not updated within four seconds.

In the simulation, a target sends a 32 byte acknowledgment to the CBR source once every two seconds. This acknowledgement message has the dual effect of notifying the CBR source that messages are reaching the target, but more importantly, it refreshes the path from the CBR to the target.

Each test was run for 900 simulated seconds and the results were averaged over ten consecutive runs in order to account for different network topologies.

### EVALUATION AND DISCUSSION

Three key performance metrics are evaluated: (i) *Packet Delivery Fraction*—the ratio of data packets successfully delivered to those originated; (ii) the *Average Latency*—the measure of the total end-to-end delay in delivering a packet to a destination; (iii) *Normalized Routing Load*—the ratio of the total number of packets transmitted by any node to the number of packets successfully delivered to the destination.

Figure 8 shows the Packet Delivery Fraction as a function of pause time and for different numbers of CBR sources. As can be seen from the graph, GRAd is insensitive to variable mobility—the percentage of good packets delivered remains essentially constant as the pause time changes.

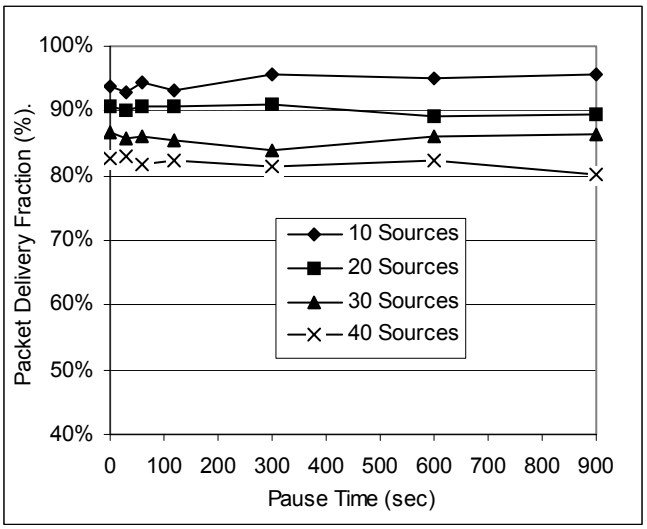



**FIGURE 8. Packet delivery fraction**

GRAd is robust in the face of changing topology because it enlists multiple neighboring nodes to relay messages from one place to another. If one node moves out of place, other nodes are often available to relay the packet without resorting to rebuilding the route.

Figure 9 shows the average end-to-end latency for successfully delivered packets. In GRAd, latency remains under 7 milliseconds, even under conditions of high mobility and load. By contrast, [Das 2000] reports end-to-end delays of more than 100 milliseconds, and as high as one second for heavily loaded networks. It must be pointed out that is not an exact comparison: in [Das 2000], packet size was 512 bytes and the radio is simulated using a different path loss model.

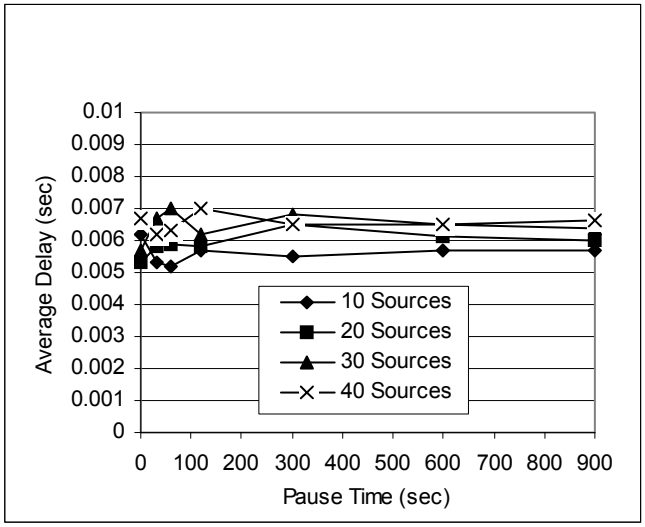



**FIGURE 9. Average delay**

In any practical implementation, GRAd is still likely to show small latencies since it avoids the RTS/CTS link to link handshake used in other protocols.

However, there is a price to pay for the "fast and loose" routing approach used by GRAd. Fig. 5 shows the routing load for various pause times and CBR sources.

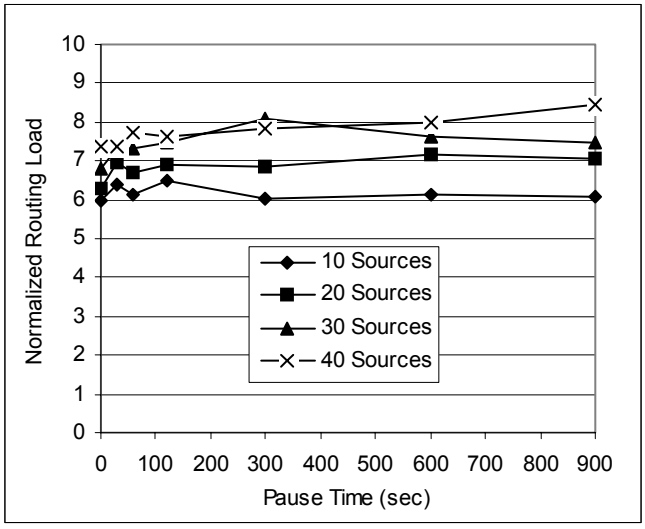



**FIGURE 10. Routing load**

As Figure 10 shows, for every one data packet received at the ultimate destination, between six and eight packets have been transmitted. Some of these "extra" packets are inevitable, for example, if a path requires two hops from source to destination, this will be recorded as a routing load of two. A destination node always sends an implicit acknowledgment notification, as described in the section on "Implicit Acknowledgement," which contributes to the load. Relatively few of the overhead packets are Reply Request messages, even in scenarios with high mobility[2]. The majority of the overhead packets are due to multiple neighbors attempting to relay the same packet. A consequence of this overhead is that GRAd networks exhibit more congestion than other network algorithms for the same offered load.

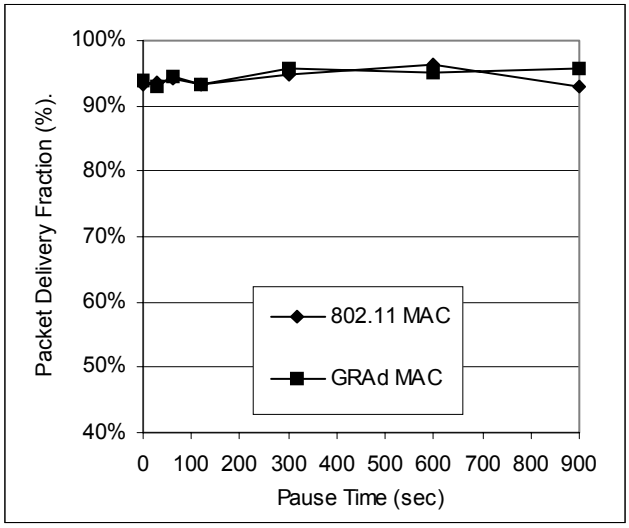
2. In a typical test with 10 sources and 0 second pause time, only 2.5% of all messages transmitted were `M_REQUEST` messages.

### CHOICE OF MAC LAYER

In any ad hoc network, there is no centralized control to control access to the airwaves, so nodes depend upon the MAC mechanism to cooperatively share the airwaves. GRAd, in particular, taxes the MAC layer since multiple neighboring nodes will attempt to relay a message soon after receiving it. It was therefore suspected that performance of GRAd would be sensitive to the choice of MAC layer.

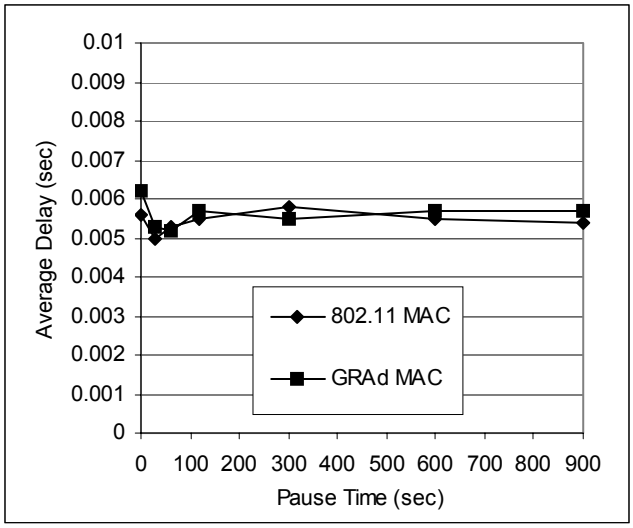
The 802.11 MAC layer [IEEE 1999] is considerably more "fair" than GRAd's simple carrier sense and exponential back off approach described in the section on "Simulation Environment." In the 802.11 approach, the MAC layer implements a countdown timer which is initialized to a random duration proportional to an exponential back off constant. The timer counts down only when the local airwaves are clear. When the timer expires, the MAC layer transmits the packet. This approach distributes air time evenly among neighboring nodes.

The tests were run for 10 CBR sources using the 802.11 MAC layer and compared against the same tests using the GRAd MAC layer—the results are shown in Figure 11. It is interesting to note that there was little effect on the overall performance of GRAd using two substantially different MAC layers.

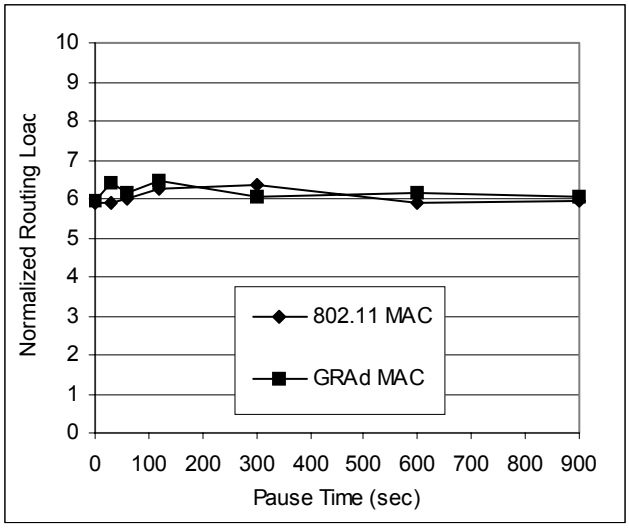



(a) Packet Delivery Fraction (10 Sources)

**FIGURE 11. GRAd vs. 802.11 MAC**




(b) Average Delay (10 Sources)




(c) Normalized Routing Load (10 Sources)

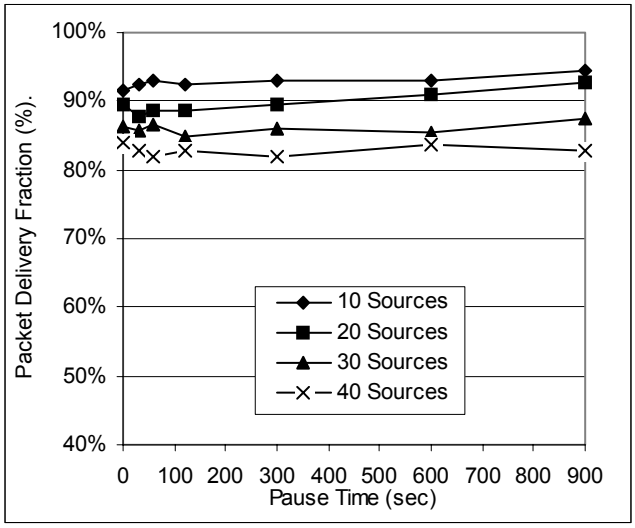
**FIGURE 11. GRAd vs. 802.11 MAC**

### DISABLING ROUTE REPAIR

A previous section describes GRAd's mechanism for Route Repair: if the receiver fails to receive a packet from a sender within the expected period of time, it sends a reply to the sender with an increased `remaining_cost`.

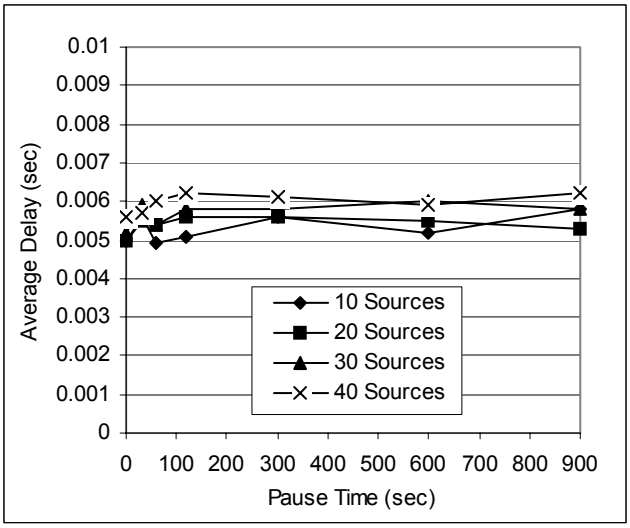
To gain insights to the effectiveness of the route repair mechanism, the tests were run with route repairs disabled: if the network topology changed so an originator no longer reached its destination (more accurately, if an originator stopped receiving packets from its destination), the entries in the cost table would time out and the originator would start a new Reply Request process.

Figure 12 shows the effects of disabling the Route Repair mechanism. The packet delivery fraction drops almost insignificantly, but somewhat surprisingly, the average latency and routing load are both improved.

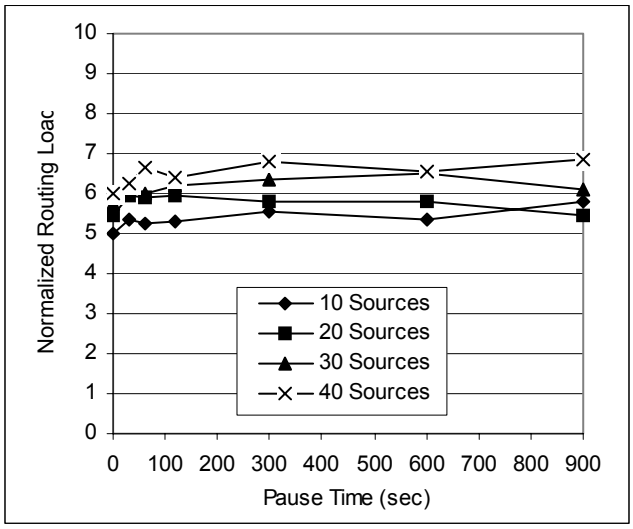



(a) Packet Delivery Fraction

**FIGURE 12. Disabling Route Repair**




(b) Average Delay




(c) Normalized Routing Load

**FIGURE 12. Disabling Route Repair**

## *Proposed extensions to GRAd*

The results of GRAd are encouraging, but there are a number unanswered questions whose answers may give insights to the operation of GRAd and suggest areas for improvement.

### CONSIDERING MORE THAN NUMBER OF HOPS

Throughout this chapter, the term "cost" has been used to mean "number of hops," but metrics other than the number of hops are possible. For example, a node can charge a higher cost for relaying a message if it notices that its local airwaves are becoming congested, or if its local topology is changing rapidly. The higher relay cost will cause messages to flow around the node if there are other nodes that can relay at a lower cost.

To generalize, in a multi-hop wireless network, the only real choice a node can make is whether or not to relay a message that it has received, and if so, when to relay it. As suggested in [Kramer 1999], it may be useful to structure the problem of routing as a set of software agents residing in the nodes and in the messages; the agents decide what should be relayed and when. The network can then be viewed as a series of *activation* and *inhibition* functions, the former causing a message to be transmitted, the latter preventing it [Intanagonwiwat 2000].

### PREFERRED NEIGHBORS

GRAd and AODV share many traits, including on-demand route discovery and updating reverse path information as a message is relayed from one node to next. GRAd permits any neighbor to participate in relaying a message, AODV insists upon a particular neighbor. A compromise between these two approaches shows some promise: A relaying node advertises a cost for relaying a message (a la GRAd) and suggests a preferred neighbor to do the relaying (a la AODV). If that neighbor is not observed to relay the message within a certain amount of time, then non-preferred neighbors may attempt to relay the message. This approach could

reduce some of the routing overhead observed in GRAd while still maintaining robustness in dynamically changing networks.

#### FUNCTIONAL ADDRESSING

The broadcast nature of GRAd's Reply Request encourages *functional addressing*, in which a node initiates an `M_REQUEST` message containing a predicate rather than a specifying a fixed target ID. The predicate is a piece of software that embodies a query such as "Are you a color printer?," "Are you a gateway to a wired network?," or "Are you an ARP server?" Each receiving node evaluates the predicate and sends a reply to the requestor if the predicate evaluates to be true. If the requestor receives multiple replies, it can choose the reply that offers the lowest `accrued_cost` (i.e. is topologically closest) or that best satisfies some other application specific criteria.

#### PER-SESSION ADDRESSING

In GRAd, routes are created on demand, entries in cost tables are short lived and persist only for the duration of a dialog between two nodes. The identities of the intermediate nodes are not required for passing messages. This opens the possibility of *per-session addressing*, in which an originating and replying nodes choose network IDs at random to be used for the duration of a session.

The space of IDs can be made large enough so the chance of two nodes choosing the same ID is insignificant.

Per-session addressing offers two advantages. The first is security: by changing its advertised address for each session, a node gains some measure of anonymity and protection against malicious eavesdroppers.

Second, manufacturing costs are reduced since network IDs don't need to be assigned and individually burned in at the time of manufacturing.

## *Summary*

GRAd offers a new approach to ad hoc, on-demand routing. Rather than sending unicast packets, it exploits local broadcasting to contact multiple neighboring nodes. Messages descend a cost gradient from originator to destination without needing to identify individual intermediate nodes. Cost functions are updated opportunistically as messages are passed from one node to the next.

Through simulation, the performance of GRAd has been tested and characterized under a variety of load and mobility conditions. The results of the tests show that GRAd exhibits very low end-to-end packet delays and offers good immunity to rapidly changing topologies.

# CHAPTER 5 *Distributed Synchronization*

In a decentralized multi-hop network, it is often desirable to distribute shared information among all the nodes in the network. Since each node can communicate with only a subset of the rest of the network, information must propagate in multiple hops if it is to reach all of the nodes in the network.

Of particular interest are the dynamics of attaining synchronization across a network of nodes. Using today's technology, it is unreasonable to assume that nodes will be fabricated with permanent real-time clocks or will have access to a common wireless time base, such as Global Positioning System (GPS) timing information. Consequently, timing must be agreed upon dynamically.

The algorithm for distributed synchronization is simple: once every $T_s$ seconds[1], node *n* broadcasts its internal time value, $\Psi_n$, to its neighbors. Upon receiving a message from a neighbor, a node adjusts its internal time to the average of its previous time and the time advertised in the message.

Given a network of N co-located nodes in which every node can receive the transmissions of all other nodes, it is easy to show that the maximum spread of the shared time will decrease by a factor of two each time a node broadcasts its value, or a factor of $2^N$ every $T_s$ seconds.

Predicting the rate of convergence for networks that are not co-located is not so simple. The maximum error among the nodes depends on both initial conditions

1. In practice, the $T_s$ interval will be randomized slightly to reduce the chance of repeated collisions among neighboring nodes.

and the order in which the nodes advertise their times, so a strict analytical approach is difficult. But using statistical models, it is straightforward to determine an upper bound for the rate of convergence.

In modeling the rate of convergence, the single most important parameter is the "diameter" of the network, where the diameter is defined to be the number of hops in the shortest path between the furthest pair of nodes. The worst case for convergence is when the nodes are arranged in a linear array with the node at one end of the array initialized to a maximum value (assumed in these tests to be 1.0) and all other nodes set to 0.0:

$$\Psi_n = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases} \quad \textbf{(EQ 7)}$$

A linear network of diameter 6 is shown in Figure 13, below.

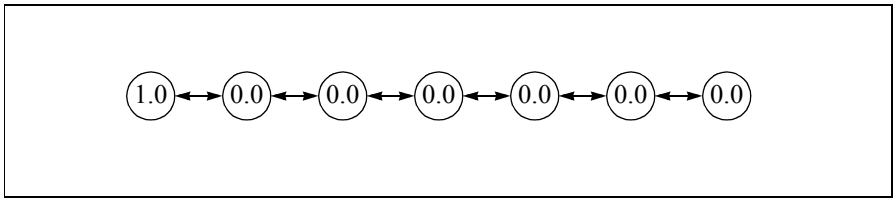



**FIGURE 13. Linear network, diameter=6**

## *Running the algorithm*

At the beginning of each trial run, a random permutation P(n) on the set of integers 0...N-1 is generated. This determines the order in which nodes broadcast.

In the course of a single iteration, each node broadcasts its internal time value, $\Psi_n$, to its neighboring nodes, so that the function:

$$\begin{aligned} \Psi'_{n-1} &= (\Psi_n + \Psi_{n-1})/2 \\ \Psi'_{n+1} &= (\Psi_n + \Psi_{n+1})/2 \end{aligned} \quad \textbf{(EQ 8)}$$

is performed N times, once for each *n*.

In the results shown below, *deviation* is defined as the maximum magnitude difference from the mean of all $\Psi_n$, and *iterations* is how many iterations were required to bring the deviation below a given threshold.

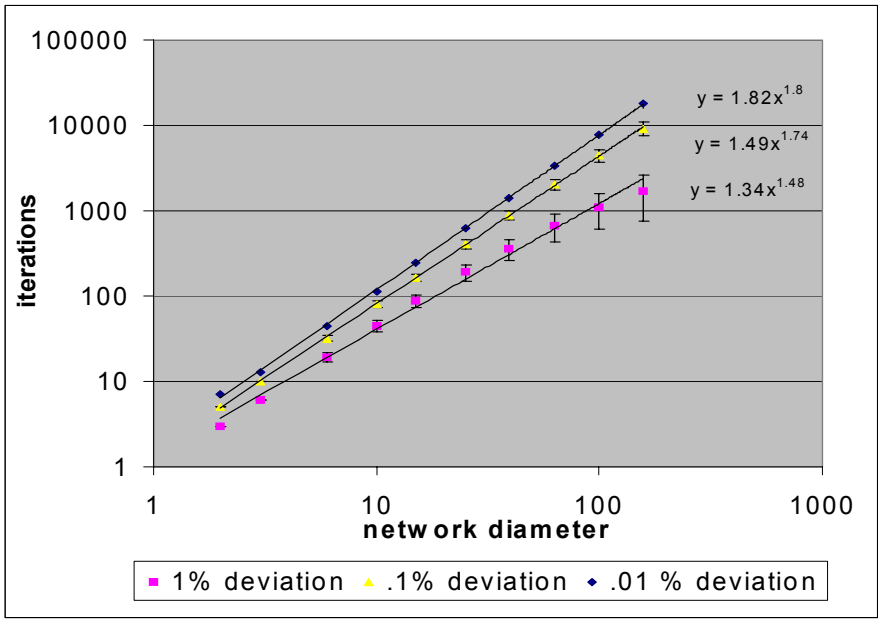



**FIGURE 14. Time to converge increases exponentially with network diameter**

Figure 14 shows that the number of iterations required to attain a given deviation increases exponentially with the diameter of the network. While theoretically troubling, this is unlikely to be a problem in practice. Both the constant and the exponent are small. To put this in context, a circular network of 1000 nodes and no overlap will have a diameter of 34. If every node in the network runs its algorithm once every second, the system is guaranteed to converge to within 0.01% of maximum deviation within approximately 1,000 seconds, or 17 minutes. For many applications, this is a trivially short amount of time.

Figure 15, below, shows deviation as a function of iterations for a variety of network diameters. It is reassuring to note that the deviation improves exponentially over linear time.

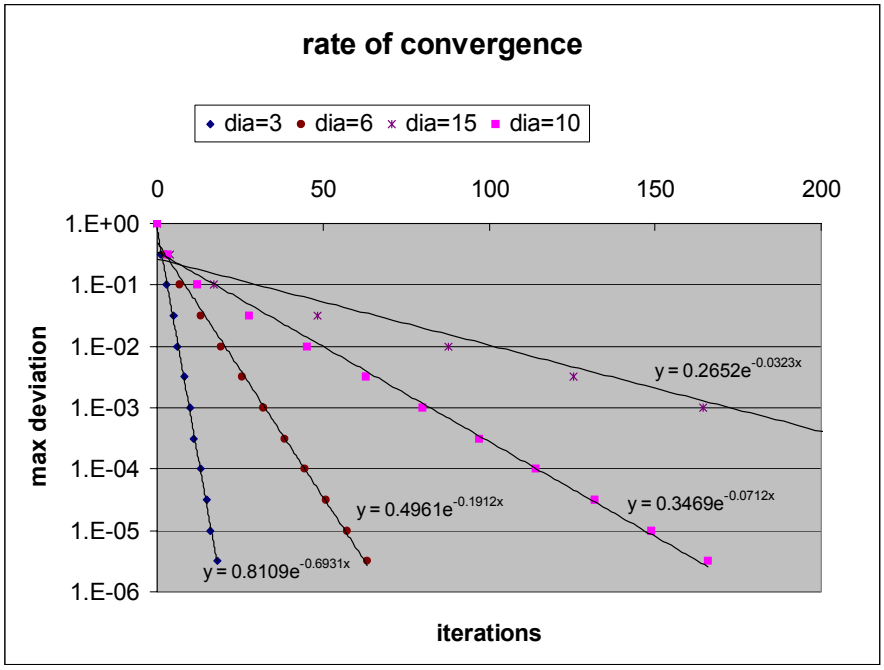



**FIGURE 15. Convergence improves exponentially at each iteration**

## *An example: synchronization for spread spectrum*

Decentralized synchronization can be used to dynamically establish a common time base among nodes in a network. A common time base can be used to coordinate the hopping sequence in spread spectrum receivers in the network, dramatically reducing the time required to attain synchronization in the receivers. The 802.11 (FH) wireless Local Area Network standard specifies a hop duration of 10 mSec. If the receivers can tolerate a 10% timing error to attain synchronization, it is sufficient to synchronize the nodes within 1 mSec of one another. The hop sequence repeats

once every 660 mSec, so synchronization within 1 part in 660, or 0.15%, is sufficient.

If 100 nodes are arranged evenly in a circular area, the resulting network will have a diameter of approximately 11. By running the simulator (or by extrapolating from the Figure 15 above) it is shown that the system will converge in under 100 iterations. From a cold start, if each node transmits a timing beacon once every second, the system will attain synchronization in under two minutes.

## *Summary*

These results show that a network can cooperatively establish a shared value where each node node periodically broadcasts its time information to its immediate neighbors and takes the average when receiving it. This approach is provably stable, and shows exponential reductions in maximum deviation in linear time for a given network diameter.

It is important to note that the figures given here are for worst-case initial conditions. Several factors can lead to substantial reductions in the time to attain a given deviation. First, perhaps contrary to intuition, mobility among nodes will tend to decrease the time required to attain a given deviation since mobility causes the errors to diffuse more rapidly.

The convergence time can be substantially reduced for nodes entering a pre-existing network simply by having the newcomers “listen but don’t talk” for a period of time. This will cause the new nodes to converge rapidly on the values that the incumbent nodes have already agreed upon.

# CHAPTER 6 *Statistical Medium Access*

## *Channel sharing*

*If it were necessary for a node to know the identities of its neighbors before communicating with them, how would the node discover their identities?*

In any network, it is often desirable to provide a communication channel to be shared among all nodes in the network. In a self-organizing network, this channel takes on special significance since it provides a mechanism for a node to share state information with its neighbors without any *a priori* knowledge of their individual identities.

Since this mechanism is used for network control and discovery, we will refer to this channel as the *control channel* through the rest of this section.

There are several ways to implement a shared channel in wireless systems, depending on the link layer communication scheme.

### A DEDICATED LINK

A node can be built with two wireless links: one to carry network control information and the other to carry data packets. This approach has several advantages. The control channel link can be implemented as a low power, low bit rate radio. Its receiver can be always on, and the rest of the system lies dormant until a message is received. Additionally, this scheme does not require nodes to be synchronized to one another.

This approach has disadvantages. The cost of the additional radios may be significant. Due to differing propagation characteristics, connectivity among control channel links may be different from connectivity among the data channel links.

#### A DEDICATED TIME SLOT

Nodes can agree on a common time slot to be used for control information. The primary advantage of this approach is cost and simplicity: no extra radio systems are required.

There is a price to pay for this approach: all nodes must be synchronized to one another. This synchronization requires that nodes are aware of one another, which is at odds with the stated design goal that the control channel is the mechanism used for nodes to discover one another.

#### A DEDICATED SPREADING CODE

For systems that use spread spectrum communication, a predetermined spreading code can be dedicated to the control channel. This has the advantage that control information can be transmitted at any time with minimal impact on data transmissions.

Using a dedicated spreading code for the control channel requires that the radio receiver in every node has two demodulators—one for the data channel and one for the control channel. This will increase the cost and power consumption of the system. If the radio systems are to attain fast synchronization at the start of each packet, the nodes themselves must be synchronized to one another.

## *Medium Access and Collision Avoidance*

Whether the control channel is implemented as a dedicated radio frequency, time slot or spreading code, it is still a shared resource and subject to contention. If mul-

tiple nodes transmit on the control channel simultaneously, there may be collisions resulting in lost information.

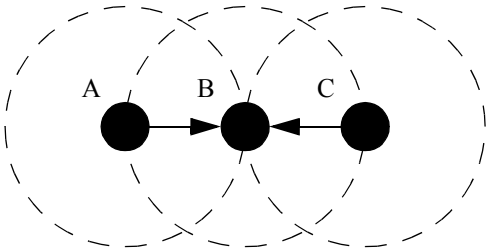



**FIGURE 16. Collision**

For example, the figure above depicts three nodes. Nodes A and C are within transmit range of node B. If both A and C transmit simultaneously on the control channel, B will not be able to discriminate between the two transmissions and the information will be corrupted, leading to lost data.

## *A statistical approach*

In order for information to be conveyed on a shared channel in a given time slot, exactly one node must transmit. If zero nodes transmit data, no information is conveyed. Similarly, if two or more nodes transmit, there is a collision, and again no information is conveyed.

In statistical channel access, every node that wishes to communicate on a shared channel does so at an agreed upon time slot, but with some probability less than one. The goal is to find the probability that optimizes the chance of successful communication, in particular, that exactly one node transmits data.

## *Choosing p*

If there is one receiving node within range of N potential transmitting nodes, and each transmitting node transmits with probability p, the likelihood of one and only one of the N nodes transmitting is given by:

$$f(p) = p(1-p)^{N-1} \quad \textbf{(EQ 9)}$$

When p is set to zero, the transmitters never transmit, so f(0) is zero. When p is set to one, the transmitters always transmit, so if there is more than one neighbor, transmissions always collide. Somewhere in between the two, f(p) rises to a maximum, depending on the number of transmitters in the vicinity of the receiver. The plot below shows the "goodput," or probability of a successful transmission for different number of transmitters as p varies from 0 to 1.

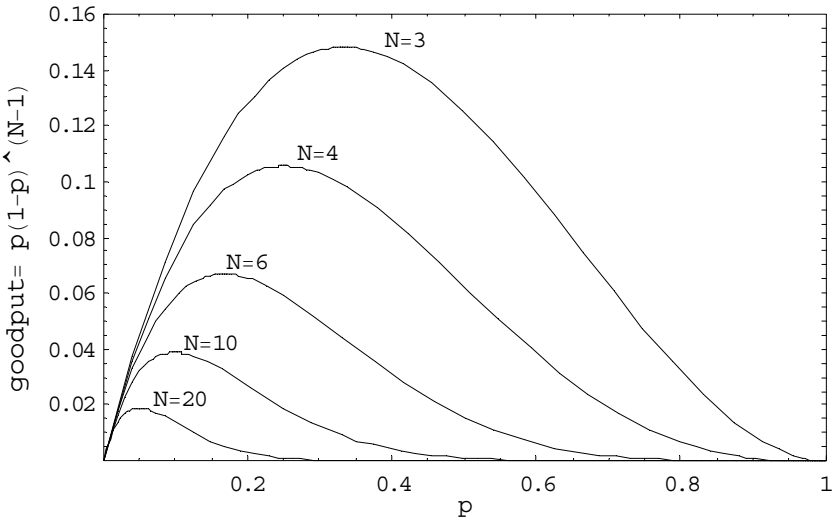



**FIGURE 17. Probability of successful transmission**

To find the value of p that maximizes the goodput, we take the derivative of **(EQ 9)** and solve for f'(p) = 0:

$$f'(p) = (1-Np)(1-p)^{N-2} = 0 \quad \textbf{(EQ 10)}$$

By inspection, it is easy to see that there are N-2 zeros when p=1 and one remaining zero when (1-Np)=0, or p=1/N.

This indicates that in order to maximize the goodput, a node that wishes to transmit to a particular receiver should do so with a probability of 1/N, where N is the number of transmitters wishing to transmit and within range of that receiver.

## *Likelihood of successful transmission*

Substituting *1/N* for *p* in equation **(EQ 9)** gives optimal goodput:

$$f(N) = \frac{(N-1)^{N-1}}{N^N} \qquad \textbf{(EQ 11)}$$

A plot of goodput versus number of nodes follows. The topmost line is the optimal goodput, attained when p is set to 1/N, where N is the number of transmitters that simultaneously wish to transmit. The two other curves correspond to p=.5(1/N) and p=2(1/N) to show the effect of non-optimal choice of p.

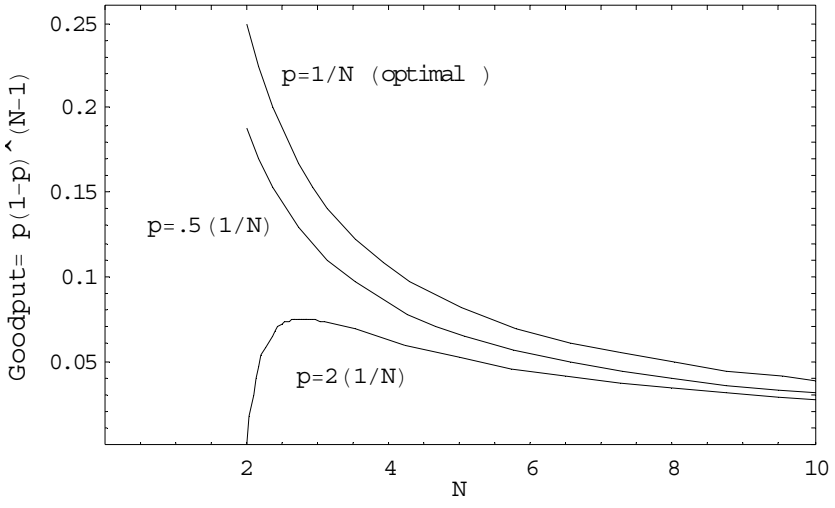



**FIGURE 18. Adjusting p as a function of the number of transmitters**

This is a function that falls off essentially as K/N, where N is the number of nodes that wish to transmit on the medium and K is 1/E, or 0.367879.

## *Statistical Medium Access in multi-hop networks*

In multi-hop networks, it is often the case that multiple nodes attempt to relay an identical message on behalf of their neighbors. In this case, if N nodes are attempting to deliver the same message, it does not matter which one of the N nodes succeed. To reflect this, **(EQ 11)** can be multiplied by a factor of N, producing:

$$f(p)N = \frac{(N-1)^{N-1}}{N^{N-1}}, p = \frac{1}{N} \qquad \textbf{(EQ 12)}$$

As the following plot shows, the probability of successful transmission converges on 1/e (0.367879) as N increases.

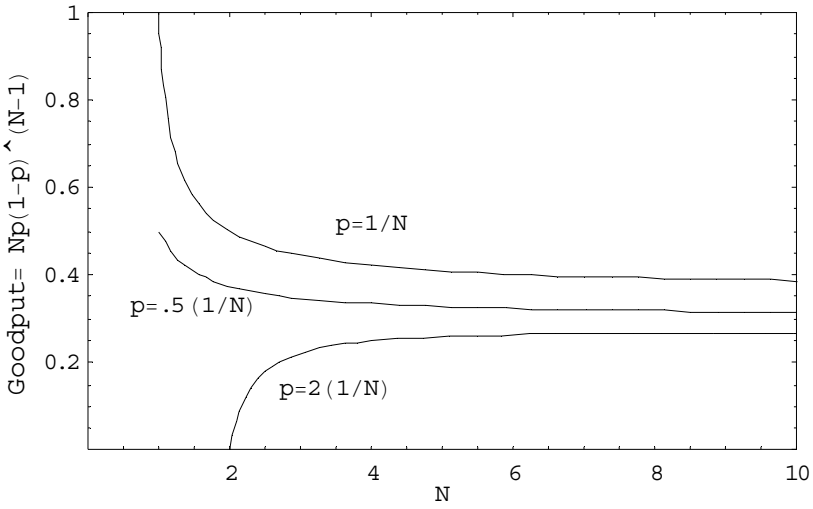



**FIGURE 19. Goodput for any of N nodes succeeding**

## *Misjudging N*

In general, it is impossible to know exactly the number of neighbors surrounding the intended recipient of a message. What happens to the overall efficiency if sending nodes over- or under-estimate the node density?

To understand the effects of non-optimal choice of p, assume a constant node density—every node has N neighbors—and that every node wants to transmit.

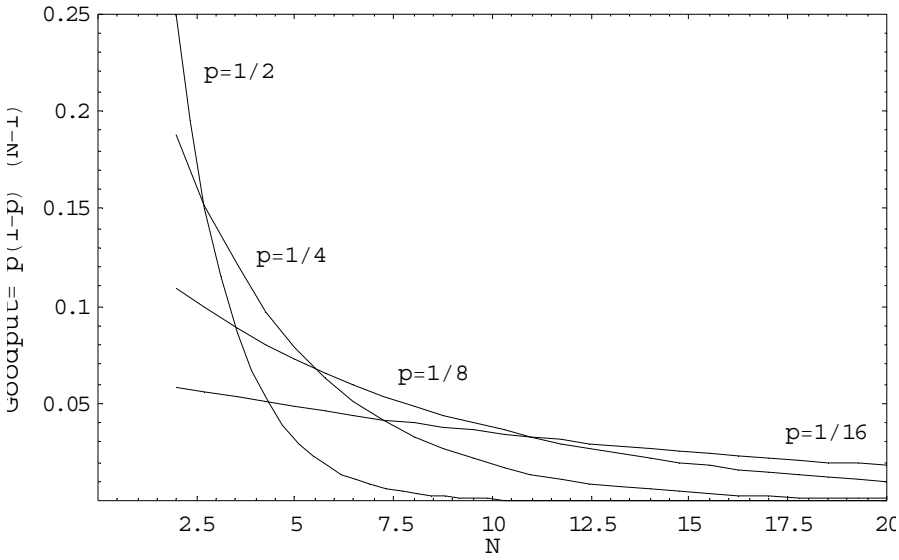



**FIGURE 20. Overestimating and underestimating p**

Figure 20 shows the likelihood of successful transmission using probabilities of 1/2, 1/4, 1/8, and 1/16. When p is chosen too large (as in underestimating N), the efficiency drops off quickly as the number of nodes increases. When p is chosen too small (as in overestimating N), the efficiency starts out lower than it would for optimal p and tapers off gradually.

Consequently, if the number of neighbors is not known precisely, it is better to overestimate N and choose a value of p smaller than optimal.

## *Summary*

Statistical Media Access can provide a simple and fair mechanism for multiple transmitters to gain access to a shared communication channel in a decentralized network. In a multi-hop wireless network where each transmitter has limited range, Shared Media Access is ideal for broadcasting information to neighboring nodes. Unlike other media access strategies, such as MACA or MACAW, it is not necessary for each transmitting node to know the identities of the receiving nodes in order to initiate a transmission. All that is required for efficient communication is an estimate of how many nearby nodes wish to transmit. When multiple nodes are attempting to convey the same piece of information, worst-case efficiency is 1/E, or 0.367879.

# CHAPTER 7 *ArborNet: A Proof of Concept*

## *Motivation*

Up to this point, this dissertation has moved in the virtual domain, using statistics and simulation to predict the behavior of Embedded Networks. But there are insights to be gained by building physical systems and reducing theory to practice. Thus it was that ArborNet was created.

ArborNet is self-organizing network consisting of twenty-five wireless nodes. Each node is housed in a small weatherproof plastic box, and contains a microcontroller, a digital radio transceiver, three AA sized batteries, a collection of sensors, Light Emitting Diodes (LEDs) and assorted "glue logic." An ArborNet node with its clear plastic cover removed is shown below in Figure 21.


**FIGURE 21. One of twenty-five ArborNet nodes**

## *Hardware system*

The heart of an ArborNet node is the “Constellation” board, designed by Andy Wheeler with assistance by the author. The Constellation board integrates a versatile 8-bit microcontroller with a short-range radio transceiver, and has proven itself to be a sound platform for network development and testing.

A block diagram of the primary components of the Constellation board is shown below in Figure 22.

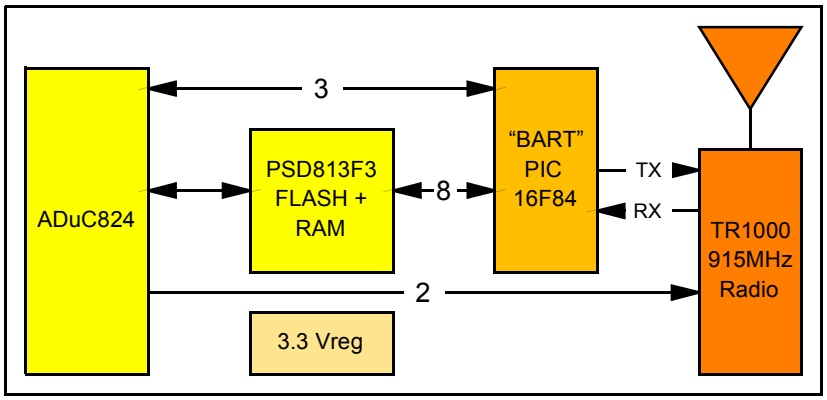



**FIGURE 22. Constellation block diagram**

### PROCESSOR

The Constellation board is built around an Analog Devices ADuC824 microcontroller. The processor is a variant of the mature 8051 family of 8-bit microcontrollers, and contains a set of features that make it especially appealing for Embedded Processing applications.

Power: the ADuC824 is a low-power device for its class. The system uses a 32KHz watch crystal as its primary system clock which is frequency multiplied via a Phase Locked Loop (PLL) to 12MHz, giving a nominal instruction cycle time of 1 microsecond with excellent power management features. The processor draws a nominal

5mA while running, but can be put to sleep, during which time the power consumption drops to approximately 5uA.

Real Time Clock: The ADuC824 has an on-board Real Time Clock (RTC), which measures years, months, days, hours, minutes, seconds with resolution down to 1/128 of a second. When the main processor is put to sleep, the RTC keeps running, so time measurements are unaffected by processor sleep times.

Digital Input/Output: The ADuC824 has a large collection of input/output lines, with support for serial I/O, I2C and ISP devices.

Analog Telemetry: The ADuC has a high-resolution A/D converter with variable gain, multiplexed inputs—a design that made it easy to add sensors to the Constellation board with a minimum of additional circuitry. The ADuC824 also has a pair of 12 bit D/A converters, though they were not used in ArborNet application.

### FLASH/RAM Expansion

Since the ADuC (and most 8051 based systems) are limited in RAM size, the Constellation includes a PSD813F memory and I/O expansion chip, manufactured by Wafer Scale Integration. The PSD813F adds 128KBytes of FLASH program memory and 2K of general purpose RAM to the system. It includes a JTAG programming port, which greatly speeds up development time during multiple revisions of the firmware.

The PSD813F also provides 32 general I/O ports. Some of these are dedicated to communication with the ADuC824, the remaining lines connect to LEDs and the BART radio interface (q.v.).

### Radio system

The Constellation board uses the TR1000 radio transceiver, a single-chip device designed by RF Monolithics. It supports bi-directional digital communications at rates up to 115K Bits/second using an unlicensed ISM frequency band of 915MHz.

The antenna is an integrated patch device made by Lynx technologies. The radio system has been measured to communicate reliably at a range of 30 meters in an open space.

### "BART" RADIO INTERFACE

The ArborNet radio communicates at a basic channel rate of 113,630 bits per second, requiring accurate 8.8 uSec timing per bit. To reduce the real-time processing requirements on the system microcontroller, the BART (Block Asynchronous Receive/Transmit) interface serves as an intermediary between the ADuC824 microcontroller and the radio. BART is implemented by a PIC16F84 microcontroller clocked at 20MHz.

To the microcontroller, the BART presents an 8-bit parallel port[1] with a 32 byte internal buffer. For the radio, the BART manages the serial data streams, providing accurate timing on transmit and byte- and bit-level framing on receive[2].

The BART provides DC balancing of the data: every 8 bits of host data is converted to 12 bits of DC balanced data upon transmission, and converted back to 8 bits upon receipt. The BART also generates and strips synchronization headers at the start of each packet, and doesn't initiate a transfer to the host microcontroller until a valid header is seen. This drastically reduces the amount of time the host microcontroller spends servicing spurious packets.

All higher level processing, including the generation and verification of per-packet CRC codes, is handled by the host microcontroller.

---

1. The BART's parallel port actually connects to the PSD expansion chip, not to the microcontroller. From a programmer's point of view, the distinction is unimportant.
2. It is worth noting that the BART chip attains bit level synchronization of the received radio bit stream to within four processor cycles, or 800 nanoseconds, without the addition of special purpose hardware.

### POWER SUPPLY

The ArborNet node is powered by three primary AA cells wired in series, which will deliver 4.5 volts when the batteries are new, drooping to under 2.4 volts over time. Since components on the Constellation board require a regulated 3.3 volts, a switching step-up/step-down voltage regulator has been included to provide a constant supply of 3.3V over the life of the batteries.

### SENSORS

The Constellation board provides inputs for one high resolution (24 bit) and one medium resolution (16 bit) analog signals. In addition, the ADuC824 has an on-chip temperature sensor, calibrated in degrees Celsius. The Constellation board also connects a battery voltage monitor to one of the analog inputs on the ADuC A/D converter, so each node can monitor and report its own battery status.

Although the Constellation boards have been tested using photo sensors and external temperature sensors, the experiments described here use only the on-chip temperature and battery voltage sensors.

### CONNECTORS

The Constellation board has number of connectors for communication and configuration, listed here.

| | |
|---|---|
| Serial I/O | 4 pins for serial input and output. Provides RX, TX, GND, +3.3V |
| JTAG Port | 14 pins. Used to program the microcontroller and PSD expansion chips. |
| PIC programmer | 6 pins. Used to program the on-board PIC (BART chip). |
| Analog In | 6 pins. Provides GND, +3.3V and two analog inputs. |
| Jumper block | 8 pins. Controls programming of the ADuC824 (_PSEN and _EA), and connects to two general purpose inputs on the ADuC824 for user configuration bits. |
| I2C/SPI | 6 pins. I2C and SPI high-speed serial interface for small peripheral devices, such as real time clocks, D/A converters, flash RAM. |

**TABLE 4. Constellation's I/O Connectors**

## *Software system*

One of the design challenges for ArborNet was fitting the software system into an eight-bit microcontroller with limited code and data storage. For the task, a small real-time kernel, “RTX51 Tiny” by Keil Software, was chosen as the basic framework for the ArborNet system.

The bulk of the ArborNet system was implemented in 3300 lines of C code over a period of four months. The development tools from Keil were easy to use. The ArborNet system made extensive use of the RTX51 thread mechanism, which resulted in code that was easy to maintain and understand. Even though the RTX51 kernel offers round robin scheduling, it was disabled in the ArborNet system to simplify the coding and eliminate the risk of race conditions. Given this conservative approach, the RTX51 kernel proved to be robust: no system errors were observed that could be ascribed to the kernel.

## *The ArborNet packet mechanism*

### SERVICES, NOT LAYERS

Classic network architectures such as the Open Systems Interconnection (OSI) networking suite defines networking as a set of layers of abstraction, providing well-defined functionality and interfaces at each layer. A layer is designed as a “black box,” hiding implementation details and communicating only its immediate super- and sub-layers. This approach is designed to simplify the implementation and testing of network systems, but in hiding information at each level from other levels, information that is required at several layers must be replicated, leading to computational and storage inefficiencies.

Embedded Networking algorithms such as GRAd thrive on “hints,” and can take advantage of all available information to increase the network efficiency. As an example, it can be useful for the MAC system in a wireless network to keep track of how many distinct neighbors are in the vicinity of the node—the MAC can use this

to predict congestion and adjust its holdoff times accordingly. In a typical layered network model, the MAC is precluded from examining any except the MAC header of received packets, so the network ID of the sending node must be included both in the MAC header as well as in the routing header.

This replication of information results in longer transmitted packets and more storage in the microcontroller. Since conserving power and storate are priorities in the design of Embedded Networking, ArborNet abandons the classic layered model in favor of a "services" oriented design.

#### LINKING, NOT ENCAPSULATION

In ArborNet, the basic unit of information transfer is a data *packet*—when the radio transmits data, it transmits a single packet. Each packet is implemented as a linked list of *segments*, where each segment carries a segment type and a payload specific to that type. The format of each segment is published and comes with a set of software functions to access the specific fields.

Consequently, any software module in the ArborNet system is allowed to examine an entire received packet for segments that it might find useful. On transmission, a packet is formed quickly and efficiently by pushing segments onto a linked list and is passed around as a single unit—no copying of memory is required as it would be for an model that uses encapsulation.

Software modules may "decorate" a packet, augmenting the information carried by the packet simply by pushing additional segments onto it. As an example, this technique was used to add networking statistics information to packets to ArborNet during system testing and debugging.

### PACKET MEMORY MANAGEMENT

The Constellation board has only 2KBytes of RAM memory, which is dominated by packet buffer storage. In this limited environment, the use of linked segments for representing packets made memory management unexpectedly efficient.

The message packet system is initialized with a pool of fixed-size segment structures, all linked into a single freelist. When any software module wishes to allocate a segment, the next available segment is simply removed from the head of the freelist. Each segment is filled in with its segment type, segment size and any appropriate data. When a software module is finished with the segment, it is pushed back onto the freelist.

The size of the fixed-length segment structure is chosen to be long enough to hold the longest segment data. Consequently, many segment types don't fill out the entire segment storage. During radio transmission, the segment is compressed by sending a single byte length field followed by the segment type and only as many bytes of the segment payload as are actually used. Upon reception, the inverse process takes place: compressed segment structures are expanded out into fixed length segments as they are received, the component segments of a packet are linked into a single list and, if the packet is observed to be free of errors, passed to other software modules for processing. After the last software module has processed the packet, the entire packet is returned to the segment freelist.

#### PACKET SEGMENT TYPES

The ArborNet system implements the following segment types.

| | |
|---|---|
| SEG_GRAD | Gradient Routing segment. Contains originating node ID, destination node ID, packet sequence number, accrued cost and remaining budget. |
| SEG_DISCO | Gradient Discovery segment, identical in content to a SEG_GRAD packet, but obeys different rules for relaying. |
| SEG_COST_L<br>SEG_COST_H | Cost Table segments, containing the contents of the originating node's cost tables. This is split into two segment types, one for the low half of the table and one for the high, because of hardware limitations on the maximum packet size. |
| SEG_STATS | Node Statistics segment used for debugging and network testing. Contains various statistics, such as the number of packets originated, number of packets received, number of packet relayed. |
| SEG_TELEM | Telemetry information. Contains the readings of the sensor array on the originating node. |
| SEG_ARQ | Automatic Retry Request segment. The packet carries with it the retry ID and a number of retries remaining before giving up. |
| SEG_ACK | Acknowledgement segment, sent in response to a SEG_ARQ. Contains the retry ID of the SEG_ARQ being acknowledged. |
| SEG_APPX | Request for Application Transmission segment. The packet names a destination node that is requesting regular updates in the form of SEG_COST_L, SEG_COST_H, SEG_TELEM, SEG_STATS and SEG_TIME packets from the receiving node. |
| SEG_PING | Ping segment. Contains node ID and local system time. Used to advertise presence and synchronization information to neighboring nodes. |
| SEG_TIME | Time and synchronization status. The packet contains the sending node's current time, the maximum timing error recently seen and the number of SEG_PING packets generated and received. |

**TABLE 5. Segment types in ArborNet**

## *Data flow in ArborNet*

The ArborNet system is implemented using a number of threads and packet queues.

Figure 23 below shows the arrangement: rounded boxes represent processing

thread, bracketed boxes represent packet queues, and lines with arrows trace the flow of packets.

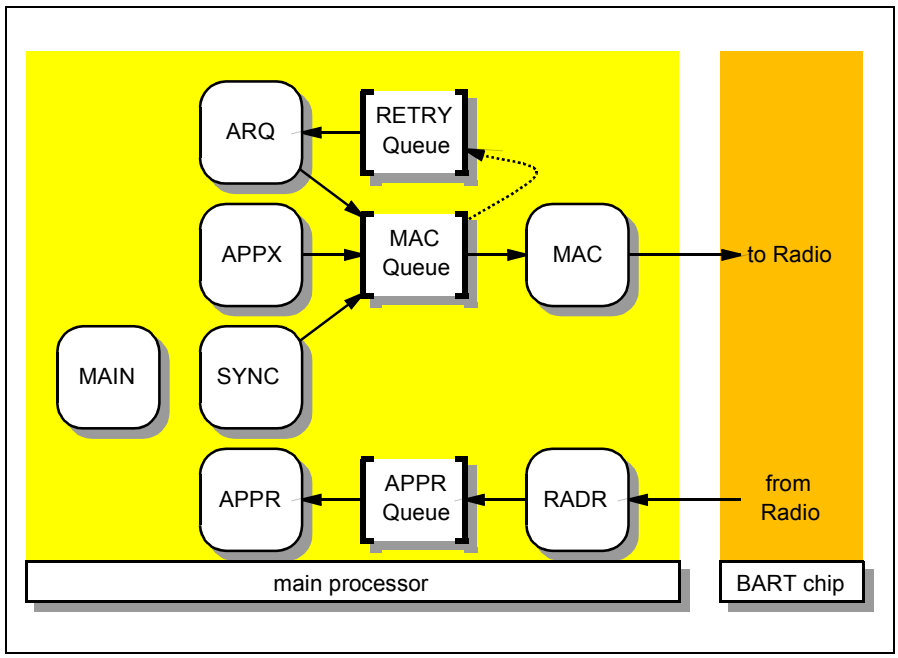



**FIGURE 23. Threads and data paths in ArborNet**

### MAIN THREAD

The Main thread handles the initialization of the system, spawning all the other threads. Once the system is running, it monitors the RS232 serial input line for commands and displays the synchronization status of the real time clock by flashing the on-board yellow LED once every two seconds.

### ARQ THREAD

The ARQ thread manages the retransmission of ARQ (automatic reply request) packets. A full description of the ARQ mechanism is described below in “ARQ processing.”

### SYNC THREAD

The Sync thread generates periodic "ping" packets that broadcast the nodes's Real Time Clock timing information to its immediate neighbors. Upon receiving a ping packet, the system adjusts its Real Time Clock as described in Chapter 5, "Distributed Synchronization." Implementation details of the synchronization mechanism are described below in "Timing services."

### APPR THREAD

The Application Receive thread monitors the APPR queue, waiting for packets received by the radio mechanism to come available. When a packet is inserted in the APPR queue, the APPR thread wakes up, removes the packet from the queue, and distributes the packet on a segment-by-segment basis to other software modules. For example, if the incoming packet contains a segment of `SEG_TYPE_PING`, it passes the packet to the synchronization system for processing.

The APPR thread also prints the contents of each incoming packet in hexadecimal form to the serial output port. This is useful for debugging[3], but is designed so any node can be a "gateway node" and log incoming packet data via the serial port.

### APPX THREAD

The Application Transmit thread is responsible for sending periodic status reports to a remote node. It waits until it receives a packet containing `SEG_TYPE_APPX`, that identifies a node wishing to receive status reports and how often those status reports should be sent. It then enters a loop, composing and transmitting cost table

3. The printing of each received packet almost certainly results in some dropped packets: Due to the non-preemptive scheduling of threads, if a new packet arrives while the system is printing another packet, the 32-byte BART FIFO can overflow, resulting in a truncated packet which will be discarded due to a CRC mismatch.

reports, analog sensor readings, synchronization status, and packet statistic packets to the requesting node.

As written, the APPX thread sends a packet on the average of once every ten seconds.

#### MAC THREAD

The Medium Access thread monitors the MAC queue for available packets to transmit. When a packet becomes available, the MAC thread delays for a random holdoff interval, using the 802.11-style exponential backoff technique described in the chapter on Gradient Routing. When the holdoff expires, the radio transmitter is set to transmit mode and the packet is passed to the BART radio interface for transmission.

#### RRCV THREAD

The Radio Receive Thread waits for an interrupt from the BART radio interface, announcing the arrival of a new packet, and proceeds to read bytes from the BART as they become available. Upon reading the end of the packet, the Radio Receive Thread verifies the packet. If the packet is valid, it is stored in the APPR queue and the APPR thread is notified of its arrival.

## *ARQ processing*

ArborNet implements a simple but effective Automatic Repeat Request (ARQ) mechanism. As described in Chapter 5, Gradient Routing works by using reverse path routing information, so sending occasional acknowledgements to an originating node is a natural and useful mechanism for keeping the routing information up to date[4].

Prior to transmission, an application may augment any packet that contains a GRAd routing segment with an ARQ segment (of type `SEG_TYPE_ARQ`), containing an

ARQ reference number and a retry count. The packet is subsequently inserted in the MAC queue for normal transmission.

When the MAC module removes a packet from its queue just prior to transmission, the packet is examined. If the packet contains an ARQ segment and the retry count of the segment is non-zero, a copy of the entire packet is installed in the ARQ's Retry Queue. The original packet is transmitted as normal.

Whenever a packet containing an Acknowledgement segment (of `SEG_TYPE_ACK`) is received, its reference number is compared against that of each ARQ segment of packets waiting in the Retry queue. If the reference number matches, the corresponding packet in the Retry queue is removed and freed—it has been acknowledged and no further retries are required.

Concurrently, the ARQ thread is run whenever a packet is installed in the Retry queue. It sets a time-out counter before attempting retransmission (typically 500 milliseconds). After the time-out expires, the ARQ thread checks to see if there is still a packet available in the Retry queue, since the queued packet may have been acknowledged and removed in the interim. If the packet is still available at the end of the timeout period, its retry count is decremented and its routing header updated before installing it in the MAC queue for subsequent transmission.

When a node receives a packet containing an ARQ segment, it responds by creating a packet with an Acknowledgement segment (of type `SEG_TYPE_ACK`) with a matching reference number and sending it to the originating node.

4. In the experiments described later in this chapter, the ARQ retry count was set to zero. This means that the recipient would generate ACK replies, but an unreceived ACK never results in retransmission of the original message.

## *Timing services*

ArborNet nodes implement the synchronization mechanism previously described in Chapter 5, “Distributed Synchronization.” This section describes the details of the implementation.

In an ArborNet node, local time is represented by an integer indicating 1/128ths of a second and is taken modulus 7680. Consequently, the system has a time resolution of 7.8125 milliseconds and cycles once every minute. These values were chosen based on the resolution of the ADuC824 Real Time Clock hardware and the limits of imposed by representing values in a sixteen bit unsigned integer. As an implementation detail, the current time is formed by reading the Real Time Clock and adding its value to an offset. When adjusting the local time, ArborNet code never explicitly sets the Real Time Clock, it only modifies the local offset.

A ping segment has the following fields:

| | |
|---|---|
| fNodeID | Node ID of the transmitting node. |
| fTimeX | Local time of the sending node. |
| fTimeR | Local time of the receiving node. |

**TABLE 6. Contents of a SEG_TYPE_PING packet**

The purpose of the SYNC thread is to broadcast the node’s local time to its immediate neighbors quasi-periodically. The thread first pauses for a randomly chosen amount of time between 0.5 and 1.5 seconds then generates a packet containing a single ping segment (of type `SEG_TYPE_PING`). Although the segment contains a structure slot for the local time (`fTimeX`), it isn’t filled in yet. The packet is installed in the MAC queue for transmission like any other packet.

The MAC contains code for special handling of `SEG_TYPE_PING` segments. At the onset of *every* transmission, the MAC code caches the local time. While the packet is being copied into the transmit buffer for processing by BART, if a segment of type `SEG_TYPE_PING` is detected, the previously cached time is written into the

fTimeX slot of the segment. This technique eliminates any timing jitter introduced by the MAC exponential backoff mechanism.

Similarly, upon receipt, the Radio Receive mechanism caches the local time when a packet first starts to arrive. In the course of reading the packet, if the Radio Receive code detects a segment of type SEG_TYPE_PING, then it copies the cached local time into the fTimeR slot. The packet is then installed in the Application Receive queue like any other packet. This technique eliminates any timing error that would result while the packet sits waiting for processing in the Application Receive queue.

When the Application Receive thread eventually dequeues the packet, it is passed to the synchronization mechanism for processing. There, the error between fTimeX and fTimeR is computed and the system clock is advanced or retarded by one half of the error.

In addition to its role as keeper of local time, the synchronization system also maintains statistics on how many ping packets were sent, how many were received, and a measure of the maximum timing error observed recently. These statistics are made available for transmission in a SEG_TYPE_TIME segment whenever the application transmit thread requests them.

## *Field tests and results*

ArborNet was subjected to field tests in two different locales. The first tests were conducted in a residential setting, for which the nodes were placed around the author's house and garden. Other tests were conducted in an office setting: the nodes were distributed around the fourth floor of the MIT Media Laboratory.

A summary of the tests are show in Table 7, below.

| Test name | Locale | Start Time | End Time | Duration | # nodes |
|---|---|---|---|---|---|
| Residential I | Residence | 01:16 | 08:04 | 7h50m | 15 |
| Residential II | Residence | 08:15 | 12:15 | 4h00m | 15 |
| Office I | Media Lab | 21:00 | 00:00 | 3h00m | 21 |

**TABLE 7. Field test overview**

In each test, node A ("Aspen") was designated as the collection point and gateway for ArborNet data. Its serial port was connected to a laptop computer which was used to log the incoming data for subsequent analysis. Tests ranged from three hours to nearly eight hours, during which time over five megabytes of raw data were collected.

The logged data consists of reports from each node in the network as it was received wirelessly at the central collection point. Reports gave a historical view of the state of each node at ten-second intervals, describing the node's cost tables, synchronization status, packet reliability statistics, on-chip temperature and system battery voltage.

## *Topology tests*

As part of the GRAd routing mechanism, each node maintains a cost table indicating the cost (or number of hops) required to relay a packet to a particular destination. This cost estimate is formed by observing how many hops were previously required to receive a packet from the destination node. Inherent in this technique is the assumption of symmetrical communication channels: if node X can receive packets from node Y, then it is assumed that node Y can receive packets from node X. In practice, radio links are not symmetrical.

Since each node reports its routing costs for all the other nodes, and since that data is collected at a single point, it is possible to derive full connectivity graphs for the network. Two such snapshots are shown below in Table 8 and Table 9. Each row

displays one node's costs, measured in hops, to the node in each column. An asterisk indicates an unknown cost. If the system has completely symmetrical links, the graph will be symmetrical around the diagonal.

A few things can be observed from these graphs. Nodes E, G, P, and U (aka Elder, Ginkgo, Pear, Sycamore, and Uri) were not active during these tests, and as the gateway, node A (Aspen) did not log reports on itself.

The smaller network deployed in the residential setting displays nearly perfect symmetry. Few of the paths require more than one hop and the physical environment didn't pose a challenge to RF communications: the paths were short and there were no significant sources of RF interference.

| S\D | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0 | | | | | | | | | | | | | | | | | |
| B | 1 | 0 | 1 | 1 | * | 1 | * | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 2 | * | 1 | 1 |
| C | 1 | 1 | 0 | 1 | * | 1 | * | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | * | 1 | 2 |
| D | 1 | 1 | 1 | 0 | * | 1 | * | 1 | 1 | 1 | 2 | 1 | 2 | 2 | 2 | * | 2 | 2 |
| E | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| F | 1 | 1 | 1 | 1 | * | 0 | * | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 2 | * | 1 | 1 |
| G | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| H | 1 | 1 | 1 | 1 | * | 1 | * | 0 | 1 | 1 | 2 | 1 | 1 | 1 | 2 | * | 1 | 1 |
| I | 1 | 1 | 1 | 1 | * | 1 | * | 1 | 0 | 1 | 2 | 1 | 1 | 1 | 2 | * | 1 | 1 |
| J | 1 | 1 | 1 | 1 | * | 1 | * | 1 | 1 | 0 | 2 | 1 | 1 | 1 | 2 | * | 1 | 1 |
| K | 2 | 1 | 1 | 2 | * | 1 | * | 1 | 2 | 2 | 0 | 1 | 1 | 1 | 2 | * | 1 | 2 |
| L | 1 | 1 | 1 | 1 | * | 1 | * | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | * | 1 | 2 |
| M | 1 | 1 | 1 | 1 | * | 1 | * | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | * | 1 | 2 |
| N | 1 | 1 | 1 | 2 | * | 1 | * | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | * | 1 | 1 |
| O | 2 | 2 | 2 | 2 | * | 2 | * | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 0 | * | 1 | 2 |
| P | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| Q | 1 | 1 | 1 | 2 | * | 1 | * | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | * | 0 | 2 |
| R | 2 | 1 | 2 | 1 | * | 1 | * | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | * | 2 | 0 |

**TABLE 8. Connectivity graph for Residential I and II tests**

The cells of the table that correspond to asymmetrical links are highlighted in gray in these tables. For clarity, only cells on the lower diagonal of the table are highlighted.

The connectivity graph for the Office I test paints a different picture, as seen in Table 9 below.

| s\d | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| B | 1 | 0 | 1 | 1 | * | 1 | * | 2 | 2 | 1 | 2 | 3 | 3 | 3 | 2 | * | 3 | 2 | * | 3 | * | 4 | 3 | 4 | 1 | 2 |
| C | 1 | 1 | 0 | 1 | * | 2 | * | 3 | 3 | 1 | 1 | 2 | 3 | 3 | 2 | * | 3 | 2 | * | 3 | * | 4 | 3 | 4 | 2 | 3 |
| D | 2 | 1 | 2 | 0 | * | 2 | * | 3 | 3 | 1 | 2 | 3 | 3 | 3 | 1 | * | 2 | 1 | * | 2 | * | 2 | 2 | 3 | 2 | 1 |
| E | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| F | 1 | 1 | 1 | 2 | * | 0 | * | 1 | 1 | 2 | 2 | 2 | 3 | 5 | 2 | * | 2 | 2 | * | 3 | * | * | 3 | 5 | 2 | 1 |
| G | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| H | 2 | 2 | 2 | * | * | 1 | * | 0 | 2 | 2 | 2 | 2 | 3 | 3 | 1 | * | 3 | 2 | * | 3 | * | * | * | * | 3 | 2 |
| I | 2 | 2 | * | * | * | 1 | * | 1 | 0 | * | * | 3 | * | 5 | 2 | * | 3 | * | * | * | * | * | * | * | 3 | 2 |
| J | 1 | 1 | 1 | 1 | * | 2 | * | 2 | 3 | 0 | 1 | 3 | 2 | 2 | 1 | * | 2 | 2 | * | 3 | * | 3 | 3 | 4 | 2 | 2 |
| K | 2 | 2 | 1 | 1 | * | 2 | * | 2 | * | 1 | 0 | 1 | 2 | 2 | 1 | * | 2 | 2 | * | 3 | * | 4 | 3 | 4 | 2 | 2 |
| L | 3 | 2 | * | 1 | * | 2 | * | 2 | 3 | 1 | 1 | 0 | 1 | 1 | 1 | * | 2 | 2 | * | 3 | * | 4 | 3 | 4 | 1 | 1 |
| M | 3 | * | * | * | * | * | * | * | * | * | 2 | 1 | 0 | 1 | 2 | * | * | * | * | * | * | * | * | 2 | 2 | 2 |
| N | 4 | * | * | * | * | * | * | * | * | * | * | * | 1 | 0 | * | * | * | * | * | * | * | * | * | 1 | * | 2 |
| O | 2 | 2 | 2 | 1 | * | 2 | * | 1 | 3 | 1 | 1 | 1 | 2 | 2 | 0 | * | 3 | 2 | * | 2 | * | 3 | 2 | 4 | 2 | 1 |
| P | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| Q | 2 | 2 | * | 1 | * | 2 | * | 2 | 3 | 2 | 2 | 2 | 3 | 4 | 1 | * | 0 | 1 | * | 2 | * | 2 | 1 | 3 | 3 | 1 |
| R | 2 | 2 | 2 | 1 | * | 2 | * | 3 | 3 | 2 | 2 | 2 | 3 | 4 | 2 | * | 1 | 0 | * | 1 | * | 1 | 1 | 2 | 3 | 1 |
| S | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| T | 3 | * | * | * | * | * | * | * | * | * | * | * | * | 5 | * | * | 2 | 1 | * | 0 | * | 1 | 2 | 1 | * | 2 |
| U | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| V | 4 | * | * | * | * | * | * | * | * | * | * | * | * | 5 | * | * | 2 | 1 | * | 1 | * | 0 | 2 | 1 | * | 2 |
| W | 3 | * | * | * | * | * | * | * | * | * | * | 3 | * | 5 | 2 | * | 1 | 1 | * | 2 | * | 2 | 0 | 3 | * | 2 |
| X | 4 | * | * | * | * | * | * | * | * | * | * | * | * | 1 | * | * | * | * | * | 1 | * | 1 | 2 | 0 | * | * |
| Y | 2 | 1 | 2 | 2 | * | 2 | * | 3 | 3 | 2 | 3 | 1 | 2 | 2 | 2 | * | 3 | 3 | * | * | * | * | * | 5 | 0 | 2 |
| Z | 2 | 2 | * | 1 | * | 1 | * | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 1 | * | 1 | 1 | * | 2 | * | 3 | 2 | 3 | 3 | 0 |

**TABLE 9. Connectivity graph for Office I test**

The network is not only larger, but the paths are longer and asymmetry is prevalent. The Media Lab is a modern office building with concrete load-bearing walls, metal doors and equipped with wireless networking gear competing in the same 915MHz frequency band as the ArborNet transceivers.

A diagram of the physical layout of the Media Laboratory and the placement of the nodes offers some additional insights to the network, shown below in Figure 24.

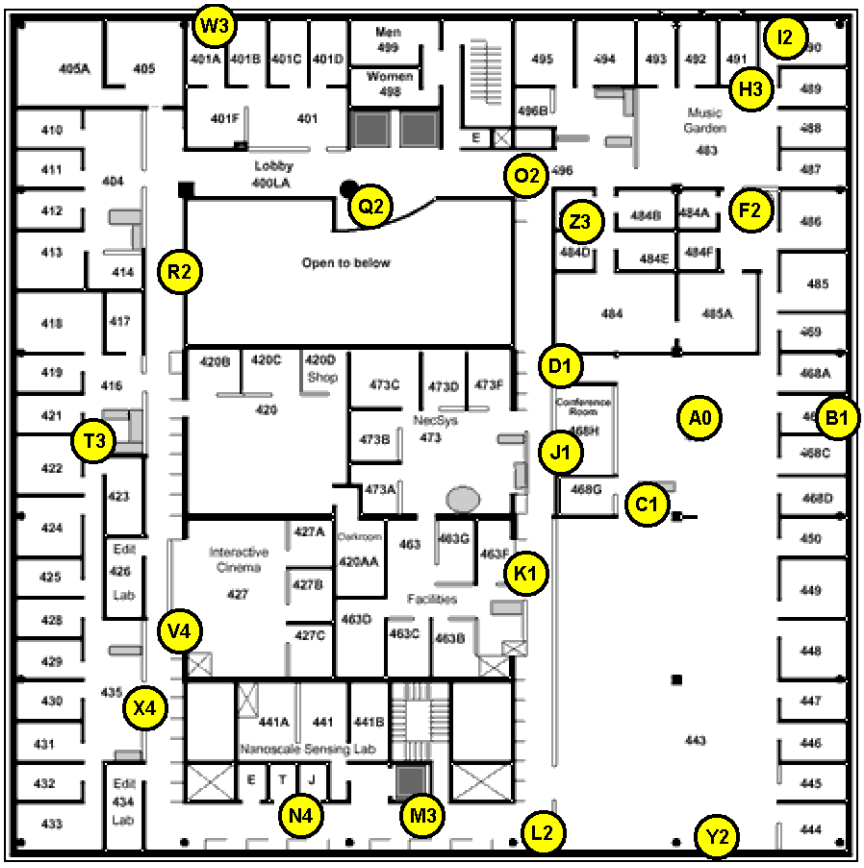



**FIGURE 24. Layout of nodes in Office I test**

Each circle represents the placement of a node. The letter is the node ID, the number is the number of hops from the central collection point (node A) located in Room 468 towards the east side of the building.

One thing to note is that physical distance is not necessarily a good indicator of the number of hops requires to relay a message. For example, node H (Holly) reported a cost of three hops to relay a message to node A, while node I (Ironwood) required only two hops, even though it was further away and on the far side of a metal door.

## *Received packet error rates*

Each node keeps statistics on packets transmitted and received and reports these statistics back to the data collection node in `SEG_TYPE_STATS` packets. One set of statistics is maintained by the radio receive process, and simply logs how many packets are received with valid CRCs and how are invalid. A rough measure of the quality of reception at each node can be had by computing

$$\frac{validPackets}{validPackets + invalidPackets} \quad \textbf{(EQ 13)}$$

This success rate for the Office I is shown for each node below in Figure 25. These figures do not account for packets with damaged synchronization headers since such packets are filtered out by the BART radio interface chip without notifying the host processor.

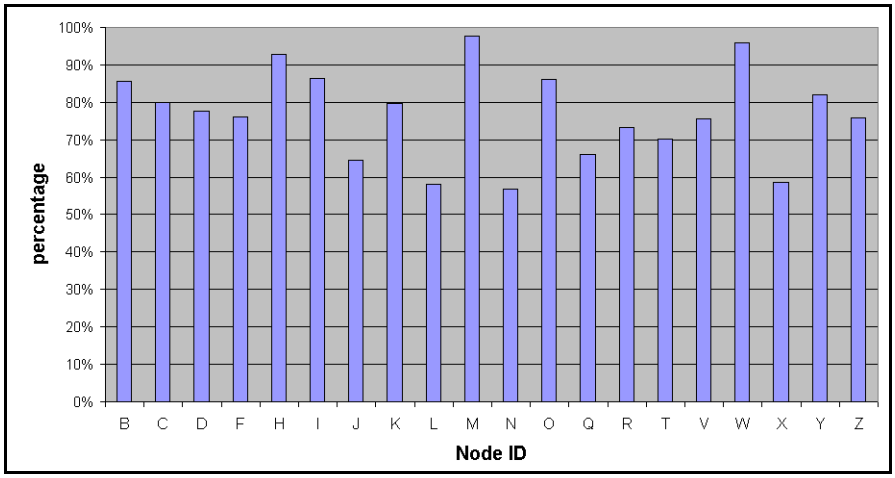



**FIGURE 25. Percentage of packets received with valid CRC**

It can be seen that nodes J, L, N, and X have marginal reception, as evidenced by their low percentage of valid packets received. Although not shown in the floor-plan, nodes L, N and X are relatively isolated and separated from other nodes by steel fireproof doors, which could account for their poor reception. It is not clear why node J has poor reception. It is located near a 915MHz wireless network access point, but so are nodes D and K, which didn't suffer from poor reception.

## *Goodput tests*

While the number of valid packets received is one way to characterize the network, it doesn't answer how successful the network is in relaying messages back to a central collection point. A more significant measure is the ratio of the number of packets originated at each node versus the number of good packets received at the collection point, or the "goodput."

Nodes were programmed to transmit a status report to the central collection node approximately once every ten seconds, so in the course of a three hour test one would expect 1062 reports from each node. An examination of the log files show that three nodes stopped transmitting before the full three hours had elapsed, so in measuring the goodput, number of transmitted nodes was prorated by the duration of the each node's lifetime. Figure 26, below, shows the goodput for each node.

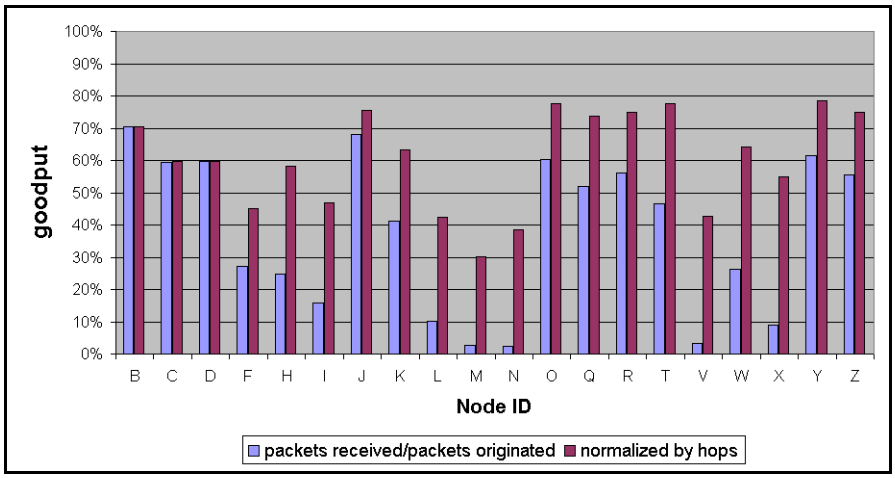



**FIGURE 26. Goodput versus node**

For each node, two values are shown: one is the goodput, which is simply the ratio of the number of packets received to the number of packets originated. The other value is the goodput normalized by the number of hops:

$$norm = goodput^{1/(hops)} \quad \textbf{(EQ 14)}$$

which is a measure of the average reliability of each link independent of the number of hops.

This goodput shows some unexplained anomalies compared to Figure 25. For instance, node J showed a high percentage of bad received packets (as seen in Figure 25), yet is among the most successful at delivering packets to the collection point (Figure 26). It is possible that there is something about the placement or even the fabrication of the node that makes its radio receiver less sensitive.

By contrast, Figure 25 showed that Node M was able to receive packets reliably, yet it shows the poorest performance in delivering packets to the collection point. Looking at the floorplan in Figure 24 offers a hint as to what might be going on. Packets from node M are relayed through node N, which is demonstrably bad at

receiving packets. It is likely that N is dropping many of the packets that M expects it to relay on its behalf.

Despite these low percentages, most of the network continued to deliver reliable data over the course of the test. Many of the techniques described in GRAd were omitted in these tests, including timing out of cost table entries and Route Repair, so higher goodput should be easily attainable.

## *Distributed temperature sensing*

*A network of simple temperature sensors can detect when a house mate is taking a shower or when a cloud passes overhead.*

Each ArborNet node is equipped with a temperature sensor incorporated into the ADuC microprocessor. The microprocessor itself consumes about 15 mW of power, so it contributes little to the overall heat of the system. When located away from direct sunlight, the air temperature inside the box is a reasonable approximation of the external air temperature, making it a useful tool for measuring the ambient temperature.

For both residential tests, ArborNet nodes were scattered indoors as well as outdoors, so a single data collection point served to measure the entire environment. A plot of the indoor temperatures, measured between 8:15 AM and 12:15 PM on a chilly Cambridge day tells an interesting story, as shown here in Figure 27.

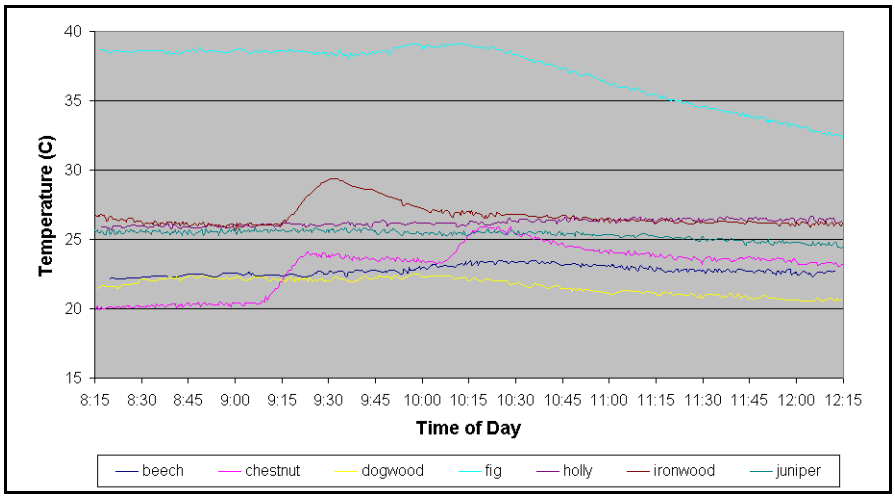



**FIGURE 27. Residential II: indoor temperatures**

The plot of indoor temperatures shows several significant features. The downstairs rooms (as measured by Beech, Chestnut and Dogwood), are approximately four degrees colder than the upstairs rooms (measured by Holly, Ironwood and Juniper). Ironwood, located in the upstairs bathroom, detected someone taking a shower at 09:15. The water and drain pipes run alongside the downstairs bathroom, causing it to warm up as well (Chestnut). Fig reports that the utility closet, containing the furnace for the baseboard heaters, is a balmy 37 degrees during the night as it struggles to keep the house warm, but cools off by more than ten degrees during the day as the rising outdoor temperatures reduce the thermal burden on the furnace.

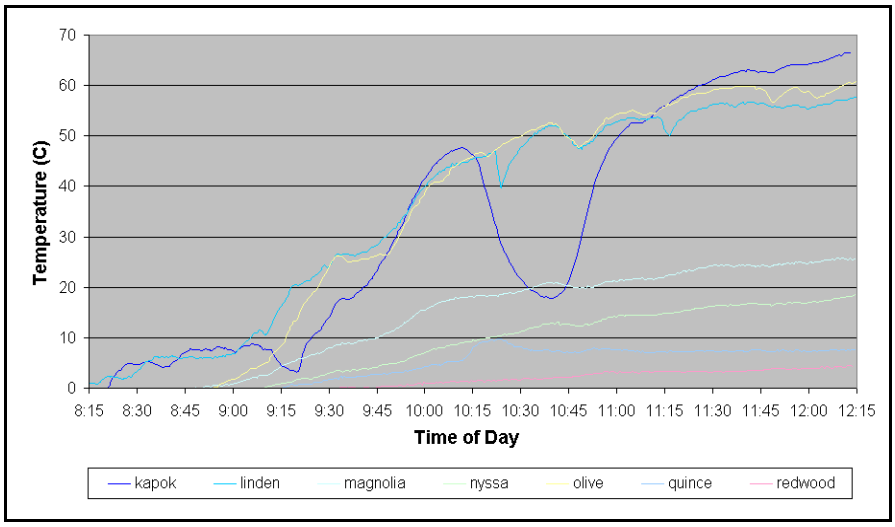



**FIGURE 28. Residential II: outdoor temperatures**

A plot of the outdoor temperatures, measured by the same network over the same period of time, shows even greater dynamics. The night air was below freezing, but temperatures climbed after sunrise. The temperatures in ArborNet node packages subjected to direct sunlight (Kapok, Linden, Magnolia) rose quickly to above 60 degrees. Kapok was located below a horizontal plank that acted as a gnomon, casting a shadow on it between 10:15 and 11:00 and causing it to cool. A cloud passed overhead around 10:50, as evidenced by a dip in temperature across all of the outdoor nodes.

It is significant that nodes equipped with something as simple as a single temperature sensor can be linked in a distributed network to glean information that would not be possible from more complicated sensors located at a single source.

A temperature graph created from the Office I test shows considerably less movement over time than the residential tests, as shown in Figure 29 below.

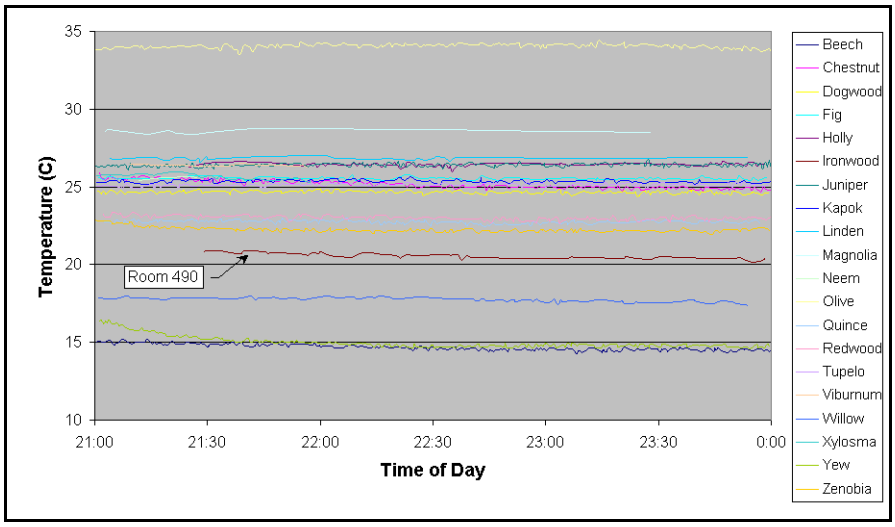



**FIGURE 29. Office I: building temperatures**

Three of the nodes (Beech, Willow, Yew) are located on metal window sills, so they register a temperature much colder than the rest of the building. Nodes Magnolia and Olive were placed on top of lighted exit signs, thus registering a considerably warmer reading. But one office, Room 490 as reported by Ironwood, clearly stands out as several degrees colder than the rest of the building.

When dense networks of sensors are located in and around office buildings, maintenance personnel can monitor large heating and air conditioning systems continuously and to a level of detail not otherwise possible, which will lead to less wasted energy and happier building occupants.

## *Battery power: trends and outliers*

As previously stated, it is important for nodes in a self-organizing network to be as autonomous as possible. In a typical network, nodes that are powered by batteries can be problematic, since it may not be clear if a loss of communication is due to the batteries running out or due to some other failure.

Nodes in ArborNet include an on-board battery monitor (measured before the voltage regulator), and are programmed to report their battery status regularly to the central data collection point. The table below shows the battery voltages in each node at the start and end of each of the three field tests, as reported wirelessly to the logging node.

| | Residential 1 | | Residential II | | Office I | |
|---|---|---|---|---|---|---|
| **Node ID** | **start** | **end** | **start** | **end** | **start** | **end** |
| B | 4.245 | 4.138 | 4.134 | 4.111 | 4.050 | 3.961 |
| C | 4.443 | 4.250 | 4.244 | 4.202 | 4.212 | 4.169 |
| D | 4.467 | 4.263 | 4.259 | 4.201 | 4.216 | 4.174 |
| F | 3.922 | 3.859 | 3.856 | 3.798 | 3.740 | 3.689 |
| H | 4.536 | 4.318 | 4.312 | 4.260 | 4.251 | 4.219 |
| I | 4.527 | 4.320 | 4.315 | 4.265 | 4.237 | 4.191 |
| J | 4.488 | 4.314 | 4.310 | 4.261 | 4.263 | 4.219 |
| K | 4.429 | 4.141 | 4.136 | 4.217 | 4.199 | 4.152 |
| L | 4.259 | 3.965 | 3.963 | 4.164 | 4.148 | 4.102 |
| M | 4.496 | 4.194 | 4.188 | 4.228 | 4.249 | 4.225 |
| N | 4.536 | 4.205 | 4.199 | 4.177 | 4.246 | 4.242 |
| O | 4.548 | 4.227 | 4.221 | 4.279 | 4.280 | 4.254 |
| Q | 4.514 | 4.189 | 4.182 | 4.140 | 4.231 | 4.188 |
| R | 4.514 | 4.192 | 4.186 | 4.111 | 4.204 | 4.165 |
| T | -na- | -na- | -na- | -na- | 4.643 | 4.488 |
| V | -na- | -na- | -na- | -na- | 4.625 | 4.578 |
| W | -na- | -na- | -na- | -na- | 4.591 | 4.423 |
| X | -na- | -na- | -na- | -na- | 4.625 | 4.565 |
| Y | -na- | -na- | -na- | -na- | 4.698 | 4.494 |
| Z | -na- | -na- | -na- | -na- | 4.684 | 4.480 |

**TABLE 10. Battery voltages before and after each field test.**

Even after a total of fifteen hours of service, the voltage in most of the batteries is still over 4.1 volts. It is clear that node B and F (Beech and Fig) were used for additional tests before the start of the field tests, and that the batteries in nodes T through Z (Tupelo through Zylosma) were fresh at the start of the Office I test.

Being able to continuously monitor the supply voltage of each node has already shown itself to be an important feature in the development and maintenance of battery powered wireless nodes. By reading many nodes at once, it is possible to detect overall trends as well as individual exceptions.

## *Synchronization*

The nodes in ArborNet implement the synchronization techniques described in Chapter 5, "*Distributed Synchronization*." To verify correct operation of the synchronization algorithm, each node issues regular reports on its synchronization state, indicating the node's local time, the maximum short-term inter-node timing difference, and the number of `SEG_TYPE_PING` packets issued and received.

The real time clock on the Constellation board has a resolution of 1/128 of a second, which sets ArborNet's limits of synchronization[5]. The synchronization status reports will only report synchronization to within 3/128 of a second, even while the internal synchronization is more accurate[6]. Consequently, the minimum reported error will be never be less than 23.4 milliseconds.

A graph of the error distribution of the synchronization reports in the Office I test indicate that nodes are synchronized within 300 milliseconds of their neighbors over 93% of the time, shown here in Figure 30.

5. This resolution could easily be improved by using software or hardware phase locked loops.

6. The short-term timing difference decays exponentially as each PING packet is received. The exponential decay is computed using simple integer arithmetic and errors less than 3 are internally truncated to zero. Fixpoint arithmetic would solve the problem handily.

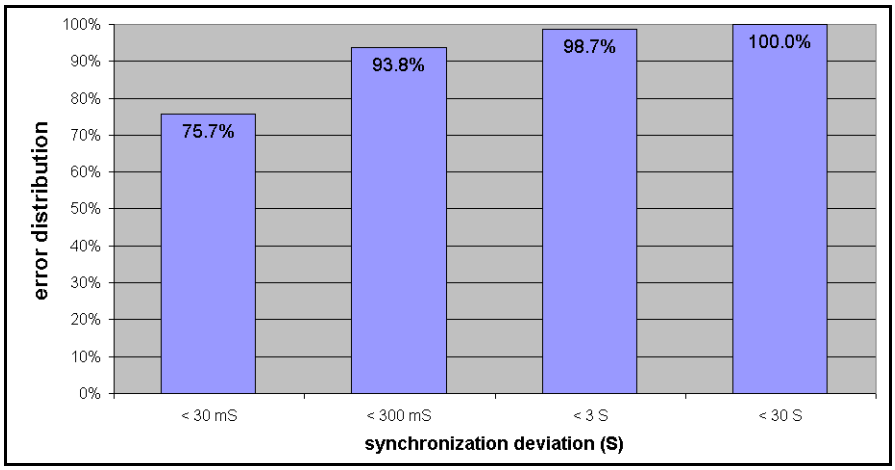



**FIGURE 30. Distribution of Synchronization Deviation**

However, that there are any errors greater than 300 milliseconds is unexpected. A snapshot of the individual synchronization errors across five nodes offers a some insights, shown below in Figure 31.

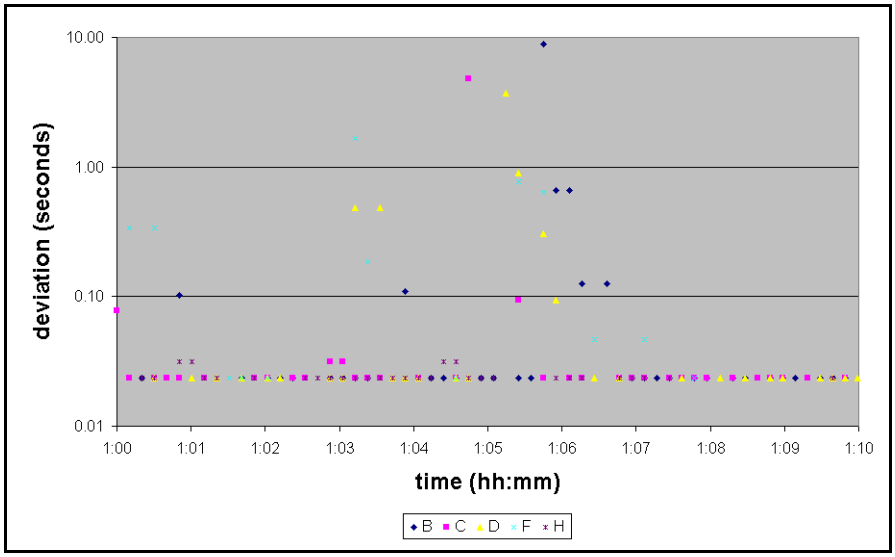



**FIGURE 31. Individual synchronization deviation (10 minute snapshot)**

Plotted on a logarithmic scale to accentuate the errors, it can be seen that the nodes report the minimum synchronization error (23.4 milliseconds) most of the time, but occasionally report errors in excess of one second. These errors do not appear to be the isolated to individual nodes; whatever the source, the errors spread like small firestorms through all the nodes in the network.

Although the log files don't capture enough information to pinpoint the source, it is possible to make some educated guesses as to the cause of the errors.

One unlikely scenario is that the real time clocks internal to the ADuC824 exhibit sufficient drift that they will fall out of synchronization after a few minutes. If a node is isolated from its neighbors for an extended period of time due to transmission errors, when it finally manages to exchange a `SEG_TYPE_PING` packet with its neighbors, its internal clock has drifted sufficiently far that it causes a ripple of synchronization error to be propagated through the network. However, the 32.768 KHz crystals used for the Constellation board's timing have an accuracy of 20 parts per million, or about 1.2 milliseconds maximum drift per minute, which doesn't explain the large timing deviations observed in the network.

A more likely cause of these errors is that nodes incorrectly exchange their internal time reference with their neighbors. For example, if the advertised synchronization information was constantly slightly behind the node's internal real time, then all the nodes in the network would retard their internal clocks as a group. However, if one node fell out of communication with other nodes, it would be free to run at the proper speed. When it re-establishes communication with other nodes, it would cause a large perturbation in the overall synchronization of the network.

A solution to this problem will require more detailed data collection and analysis. Despite these occasional timing errors, the fundamental goal has been achieved, showing that nodes can synchronize accurately to one another in a decentralized network.

# CHAPTER 8 *Conclusions & Future Work*

> *Because it has no other means to communicate, a smoke detector in the basement can only scream when it detects smoke and beep futilely when its batteries run low. Given a more sensible means of expression, it could do a much better job of protecting a home and its occupants.*

Microprocessor chips have already reached the point where their usefulness is not limited by their processing power, but rather by a lack of context. Isolated from the rest of the world, the majority of these chips can neither sense the world around them nor participate in any meaningful discourse with other chips in their environment.

## *Some lessons learned*

This thesis has presented *Embedded Networking*, an integrated approach to scalable, self-organizing networks designed to give voice to microprocessors. Several good and surprising results have arisen in the course of developing this art.

*Embedded Networks are attainable*. Embedded Networks can be built today. A practical implementation does not hinge upon as-yet-undeveloped technologies or exotic components. Because Embedded Networking has been designed to be "radio agnostic," it can use existing radios and still take advantage of the inevitable developments in new wireless technologies.

*Data aggregation is a powerful tool*. It is astonishing how much can be learned by having multiple data points. As an example, the interplay of outdoor temperatures, measured at just seven different points during the course of a sunrise, told a much more interesting story than seven readings from one sensor possibly could.

*Embedded Networks must be proactive.* David Tennenhouse is right: if computing systems are to become useful, they must do so with a minimum of attention from their human stewards [Tennenhouse 2000]. For example, it proved to be remarkably useful to have each node in the ArborNet proof of concept system monitor its own battery voltage. This simple approach removed doubts as to whether nodes were running low on power or not. As an unexpected benefit, it proved very easy to answer the question "do alkaline batteries run down faster when they are subjected to sub-freezing temperatures?" (The answer was that they did not drain appreciably faster than their warmer neighbors.)

*Gradient Routing works in theory and in practice*. Gradient Routing, a cornerstone of the Embedded Networking systems described here, proved itself to be an effective technique. It succeeded in relaying data packets from one wireless node to another without either the need for preplanning the network or for human intervention during its operation.

## *Unturned Stones*

Paradoxically, the hallmark of satisfying research is that it leaves one hungry to do more, and the work here has been no exception. This early exploration into the theory and practice Embedded Networking has perhaps raised one new question for every question answered. A few of these "unturned stones" are offered with the thought that they might prove to be interesting and worthwhile avenues for further research.

### DEEPEN UNDERSTANDING OF RADIO PROPAGATION

It would be informative to conduct a detailed study of the connectivity characteristics among all nodes in a network, directly measuring the bit error and packet error characteristics between each combination of nodes. Although the error rates will depend upon radio technology and environment, some other questions will fall out as a direct consequence: How symmetrical are wireless communication links in

practice? What is a good estimate of path loss, and how well does spatial division multiplexing work? How closely does physical topography correspond to network topology? This kind of information is typically difficult to gather, but a decentralized multi-hop wireless network such as ArborNet makes it quite easy.

#### BUILD A WIRELESS SUNDIAL

Chapter 5 described techniques for a community of nodes to agree on a common time base, appropriate for sub-millisecond measurements, but not rooted in any physical time base. Working with ArborNet suggests a somewhat whimsical approach to accurate timekeeping in an unattended network by building a "wireless sundial" from multiple embedded networking nodes.

Each node is powered entirely by solar cells, so it would only wake up when there was sun to measure. Once awake, a node would track the position of a shadow cast by a gnomon using simple photocells, and report the position of the shadow to its neighbors. Using multiple nodes would eliminate errors due to clouds, and on clear days, the network could accurately report both the solar time of day and the day of the year. The system would be guaranteed to be free of any long-term drift.

#### IMPLEMENT DYNAMIC DUTY CYCLE

Distributed Synchronization is a first step towards power savings. If all nodes in the network can agree on a common time base, they can all sleep at the same time and wake up simultaneously in order to exchange packets during network "business hours."

Assume that nodes draw no power while they sleep and constant power while they are awake, independent of radio activity. Let $T_{wake}$ denote the amount of time that the they are awake and $T_{sleep}$ the time during which they sleep, the duty cycle for the system is then:

$$T_{DC} = \frac{T_{wake}}{T_{wake} + T_{sleep}} \quad \textbf{(EQ 15)}$$

The system power consumed will be reduced by a factor of $T_{DC}$, while the load on the airwaves will be increase by the same factor.

Since many embedded network applications need only communicate to for a few milliseconds out of every minute, this approach can lead to substantial power savings.

#### DEVELOP MECHANISMS FOR RELOADING CODE

For all its ease in measuring and gathering data, the Embedded Networks presented here don't offer any mechanism for reloading code over the network. Part of this is due to limitations in hardware: the Constellation boards used in the ArborNet system have no provisions for dynamically reloading code. But a degree of caution influenced the design: one bad byte distributed among all the nodes could immediately bring down the network.

Nonetheless, the value in being able to dynamically reload code in order to conduct different networking tests is obvious. An embedded network system designed to dynamically update its own networking code would be an extremely useful research tool. A longer-term goal would be to create robust mechanisms for dynamically reloading code for applications outside of the laboratory.

## *Acknowledgements*

Entering the Media Laboratory is like embarking on some strange and wonderful journey: When I started five years ago, I didn't know where it would lead me, but I had a hunch that I would have many adventures and encounter some wonderful people along the way. Time has validated my intuition, and in retrospect, I could not have scripted a better cast of characters.

My advisor, Mike Hawley, jarred me loose from my everyday life and into the Lab in the first place, and it is his ongoing vision of building smart, useful objects that has kept me happily working late nights. Committee member and sailing captain Andy Lippman has always offered good criticisms of my work, backed up with sound reasoning. Bill Kaiser, expert in the field of self-organizing networks, has always expressed enthusiasm about my work, sensibly tempering my elation by introducing me to work other people have already done in the field. I've had many stimulating talks with LCS professor Hari Balakrishnan and his students about the finer points of embedded networks—it is wonderful to have a local expert in the field. David Tennenhouse did me a great service by making me promise that I would stay focussed on completing the dissertation before being distracted by the Next Big Thing.

I have been fortunate to have been supported as a Motorola Fellow for much of my time as a Media Lab student. But the support hasn't been simply financial: I've been constantly inspired by my interactions with the engineers and managers of Motorola, and I especially appreciate Sheila Griffin's handling of the program.

I feel particularly lucky to be part of Hawley's Personal Information Architecture team. Past members Maria Redin, Manish Tuteja and John Underkoffler have remained great friends and helped me through the inevitable bumps along the road to a degree. Current colleagues Chris Newell, Roshan Baliga and especially Paul Pham have spent large blocks of their time making ArborNet into a reality, and I have come to depend upon Bill Butera for seeing technical holes that I have missed.

Two people deserve special mention, without whose help I cannot imagine this research coming to fruition. Andy Wheeler designed the Constellation board and many other elegant (though often unsung) hardware systems. Through his own example, Andy has pushed me to think harder and build more. Charlotte Burgess advanced this research in more ways than can be counted, from proofreading and graphic design to emotional and moral support. Charlotte has a talent for asking the question that unties whatever Gordian knot I am struggling with. To both Andy and Charlotte, I give special thanks.

# APPENDIX A *References*

[Abelson 1995] Hal Abelson, Tom Knight, Gerald Sussman. "Amorphous Computing." MIT LCS internal White Paper. Draft of October 14, 1995.

[Abramson 1985] Norman Abramson. "Development or the ALOHANET" IEEE Transactions on Information Theory, Vol. IT-31, pp. 119-123, March 1985.

[Bertsekas 1992] Dimitri Bertsekas, Robert Gallager. "Data Networks" Second Edition. Prentice Hall Englewood Cliffs, New Jersey, 1992. *A dependable textbook on the networking and queuing theory.*

[Bluetooth 1999] Bluetooth specification, available online at http://www.bluetooth.com/developer/specification/specification.asp

[Broch 1998] J. Broch, D. A. Maltz, D. B. Johnson, Y-C. Hu, J. Jetcheva. "A performance comparison of multi-hop wireless ad hoc networking protocols," in *Proceedings of the 4th International Conference on Mobile Computing and Networking (ACM MOBICOM '98)*, pp. 85–97, October 1998.

[Bult 1996] K. Bult, A. Burstein, D. Chang, M. Dong, M. Dielding, E. Kruglick, J. Ho, F. Lin, T. H. Lin, W. J. Kaiser, R. Mukai, P. Nelson, F. L. Newburg, K. S. J. Pister, G. Pottie, H. Sanchez, O. M. Stafsuff, K. B. Tan, C. M. Ward, G. Yng, S. Xue, H. Marcy, J. Yao. "Wireless Integrated Microsensors." Proceedings of the 1996 Hilton Head Transducers Conference, June 1996. *An early paper on Bill Kaiser's excellent WINS program.*

[Carvey 1996] Phillip Carvey. "BodyLAN." IEEE Circuits and Devices, V4 No 12 July 1996. *Describes the design for Phil Carvey's ultra low-power radio design—a*

*nifty bag of tricks that attains $4x10^{-9}$ Joules per received bit and $2x10^{-9}$ Joules per transmitted bit.*

[Das 2000] S. Das, C. Perkins, E. Royer. "Performance comparisons of two on-demand routing protocols for ad hoc networks," *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Tel Aviv, Israel, March 2000, pp. 3–12.

[Demers 1994] Alan Demers, Scott Elrod, Christopher Kantarkiev, Edward Richley. "A Nano-Cellular Local Area Network Using Near-Field RF Coupling." CSL-94-8 Xerox Corporation, Palo Alto Research Center. October 1994. *Describes a nobel radio system developed at Xerox PARC as part of their Ubiquitous Computing program. They essentially use inductively coupled systems and exploit the rapid falloff for high-density wireless systems.*

[emWare 2000] Documentation available online at http://www.emware.com/solutions/emit/. *emWare makes a suite of "thin clients" that run on small microcontrollers connected (generally via wired links) to larger systems that act as proxies.*

[Fall 1998] K. Fall and K. Varadhan (editors) "ns Notes and Documentation." Lawrence Berkeley Nantional Laboratories, August 1998. Available online at http://wwwmash.cs.berkeley.edu/ns/. *ns has become is the industry standard for quantifying the performance of network protocols.*

[Gershenfeld 1999] N. Geshenfeld. "When Things Start To Think." Henry Holt and Company, New York, 1999. *Articulates the vision of Things That Think.*

[Heidegger 1968] Martin Heidegger (translators John Macquarrie and Edward Robinson). "Sein und Zeit." Harper & Row, San Francisco. 1962. *This famous existentialist starts with first principals in describing the relation of humans to the world around them. As such, he has a greater influence in the topic of user interfaces than some might imagine.*

[Heinzelman 2000] W. Rabiner Heinzelman, A. Chandrakasan, and H. Balakrishnan "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," Proceedings of the 33rd International Conference on System Sciences (HICSS '00), January 2000. *This paper describes LEACH (Low Energy Adaptive Clustering Hierarchy), a technique that dynamically chooses local cluster heads in multi-hop, ad hoc networks in order to balance and reduce the total amount of energy spent by individual nodes.*

[IEEE 1999] IEEE Std. 802.11, 1999 Edition. "Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." IEEE Standards Association, 1999. ISBN 0-7381-1809-5 *Description of 802.11 wireless LAN standard*

[Intanagonwiwat 2000] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks." Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCOM 2000), August 2000.

[IrDA 1998] Infrared Data Association, IrDA specification, available online at http://www.irda.org

[Johansson 1999] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek. "Routing protocols for mobile ad hoc networks—a comparitive performance analysis," *Proceedings of the 5th International Conference on Mobile Computing and Networking (ACM MOBICOM '99)*, pp. 195-206, August 1999.

[Johnson 1996] D. B. Johnson, D. A. Maltz. "Dynamic source routing in ad hoc networks," in *Mobile Computing*, T. Imielinski and H. Korth, Eds., Kulwer, 1996, pp. 152-81.

[Kelly 1997] Kevin Kelly. “New Rules for the New Economy.” Wired Magazine, 5.09 pp 140-197, September 1997. *Kevin Kelly’s vision, not just of a world densely populated with smart devices, but how these devices will change the rules.*

[Kleinrock 1987] Leonard Kleinrock and John Silvester. “Spatial reuse in multihop packet radio networks.” Proceedings of the IEEE, 75(1):156-167, January 1987. *MANET was not the first foray into multi-hop packet radio systems.*

[Kramer 1999] K. H. Kramer, N. Minar, P. Maes. “Tutorial: Mobile Software Agents for Dynamic Routing,” *Mobile Computing and Communications Review (ACM SIGMOBILE)*, vol. 3, no. 2, 1999, pp. 12–16.

[Kumar 2000] S. Kumar. “Sensor Information Techology (SenseIT) Program”, described in http://www.darpa.mil/ito/research/sensit/, DARPA Information Technology Office, April 2000.

[Macker 2000] J. Macker, S. Corson. “Mobile Ad-hoc Networks (manet) Charter.” http://www.ietf.org/html.charters/manet-charter.html, February 2000

[Metcalfe 1976] Robert M. Metcalfe and David R. Boggs. “Ethernet: Distributed packet switching for local computer networks.” Communications of the ACM 19, 7 July 1976, pp 395-404. *One of the original papers on Ethernet.*

[Mills 1991] David L. Mills. “Internet time synchronization: the Network Time Protocol.” IEEE Trans. Communications COM-39, pages 1482-1493, October 1991. *Available online as* http://www.eecis.udel.edu/~mills/database/papers/trans.ps

[Minar 1999] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian, Pattie Maes. “Hive: Distributed Agents for Networking Things.” Presented at ASA/MA August 1999. Available on-line as http://www.hivecell.net/hive-asama.html

[Moravec 1998] Hans P. Moravec. Robot: mere machine to transcendent mind. Oxford University Press, November 1998. *Has an excellent graph of MIPs per dol-*

*lar. See also the online WEB pages for regularly updated numbers: http://www.frc.ri.cmu.edu/~hpm/book97/ch3/processor.list*

[Pentland 1996] Alex P. Pentland. "Smart Rooms." Scientific American, 274(4) pp. 68-76. April 1996. *At the time this article was published, Pentland made a convincing argument that computational systems of were sensorially deprived. Five years later it still holds true.*

[Perkins 1999] C. Perkins and E. Royer. "Ad-hoc on-demand distance vector routing," *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, February 1999.

[Poor 2001] R. Poor. "Jasper: a Java-based dynamic simulator for ad hoc wireless networks," unpublished.

[Simon 1969] Herb Simon. "The Sciences of the Artificial." MIT Press, Cambridge Massachusetts, 1969. *Economics meets Artificial Intelligence meets Complexity Theory, offering useful tools for thinking about problems. A wonderfully readable survey of Simon's work.*

[Stajano 1999] Frank Stajano and Ross Anderson. "The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Networks." Chapter in *Security Protocols, 7th International Workshop Proceedings, Lecture Notes in Computer Science, 1999*. Springer-Verlag Berlin Heidelberg 1999. *Describes techniques for "imprinting" in wireless devices in ad-hoc networks can be "imprinted" to describe ownership and other security attributes.*

[Sun 2000] Sun Microsystems. "JINI Architecture Specification, Version 1.1." available online at http://www.sun.com/jini/specs/

[Tennenhouse 2000] David Tennenhouse. "Proactive Computing." Communications of the ACM, May 2000, Volume 43, #5. *David makes the argument that since there are so many embedded computers, they must operate autonomously and sen-*

*sibly—humans must be able to "get out of the loop." This issue of CACM is dedicated to the topic of "Embedding The Internet."*

[UMTS 2000] Information available online at http://www.umts-forum.org. *UMTS (universal mobile telecommunication system) is being being developed by the International Telecommunications Union as the successor to GSM cellular telephone systems, featuring bandwidths between 144KBps and 2MBps.*

[Weiser 1991] Mark Weiser. "The Computer for the Twenty-First Century." Scientific American, 265(3) pp. 94-104, September 1991. *Mark Weiser's famous paper on Ubiquitous Computing. It still make good sense after all these years, and reminds us of some of the things we are trying to attain.*

[Wheeler 2000] Andy Wheeler, Aggelos Bletsas. "Energy Efficiency Study of Routing Protocols for Mobile Ad-hoc Networks" Final project paper for 6.899 Computer Networks, taught by Professor Hari Balakrishnan.

[W3C 2000] World Wide Web Consortium (W3C). "Extensible Markup Language (XML) 1.0" second version edition 6 October 2000. Available on the web at http://www.w3.org/TR/REC-xml

# APPENDIX B *ArborNet Host Code Listing*

Following is the ArborNet C source code that is executed by the ADuC824 host processor on the Constellation board. More information on the ArborNet system can be found in Chapter 7.

## File: adc.h

```
#ifndef ADC_H
#define ADC_H
// -*- Mode: C++ -*-
//
// File:        adc.h
// Description: routines to read the A/D converters
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "pkt.h"

// routines to read, packetize, print the analog to digital converter

// thermistor (external temp) on AIN1 (primary)
// photocell on AIN1 (aux)
// Battery monitor on AIN4 (aux)
// chip temperature on aux

typedef struct _adcPayload {
  // unsigned long fExtTemp;
  // unsigned int fLight;
  unsigned int fIntTemp;
  unsigned int fVBATMon;
} adcPayload;

void adc_init();

pkt_t xdata *adc_report(pkt_t xdata *next);
// create packet and fill with a fresh set of readings

#endif
```

## File: adc.c

```
// -*- Mode: C++ -*-
//
// File:        adc.c
// Description: routines to read the A/D converters
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
```

```
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "arbor.h"
#include <rtx51tny.h>           // for os_wait()...
#include <aduc824.h>            // aduc register defs
#include "constell.h"           // leds, etc
#include "adc.h"
#include <string.h>
#include <stdio.h>
#include "screen.h"

// NOTE: With the current board design, VBATMON will stay pegged at
// full scale until VBAT drops to less than 1.25V.


// ============================================================
// ============================================================
// internal routines

#define PHOTOCELL_CHANNEL 0x00  /* AIN3 */
#define VBATMON_CHANNEL 0x10    /* AIN4 */
#define CHIPTEMP_CHANNEL 0x20   /* AINTEMP */
#define AIN5_CHANNEL 0x30       /* AIN5 */

static void adc_read_secondary(unsigned char ad1con) {
  // Take a reading on a secondary ADC channel.  16 bit result is in
  // AD1H,AD1L upon returning from the routine.
  AD1CON = ad1con;
  ADMODE = BITMASK(0,0,0,1,0,0,1,0);   // secondary, single shot
  RDY1 = 0;
  while (!RDY1) {
    os_wait2(K_TMO, 1); // buzz...
  }
}

void adc_read(adcPayload xdata *ap) {
  // take a reading of the four analog sources: photocall, battery
  // monitor, chip temperature, and thermistor.  Store results in
  // adcPayload structure, passed by reference.

  // thermistor is on primary A/D...
  // AD0CON = BITMASK(0,0,0,0,0,1,1,1);    // AIN1-GND, unipolar, 2.56V
  // ADMODE = BITMASK(0,0,1,0,0,0,1,0);    // primary, single shot
  // RDY0 = 0;                 // start conversion on primary A/D
  // while (!RDY0) {
  //   os_wait2(K_TMO, 1);      // buzz...
  // }
  // ap->fExtTemp = (AD0H << 16) | (AD0M << 8) | AD0L;
  // ap->fExtTemp = AD0H;
  // ap->fExtTemp <<= 8;
  // ap->fExtTemp |= AD0M;
  // ap->fExtTemp <<= 8;
  // ap->fExtTemp |= AD0L;

  // adc_read_secondary(BITMASK(0,0,0,0,1,0,0,0) | PHOTOCELL_CHANNEL);
  // ap->fLight = (AD1H << 8) | AD1L;

  adc_read_secondary(BITMASK(0,0,0,0,1,0,0,0) | CHIPTEMP_CHANNEL);
  ap->fIntTemp = (AD1H << 8) | AD1L;

  adc_read_secondary(BITMASK(0,0,0,0,1,0,0,0) | VBATMON_CHANNEL);
  ap->fVBATMon = (AD1H << 8) | AD1L;

}

// ============================================================
// ============================================================
// published routines


void adc_init() {
  // fastest, noisiest input
  SF = 0x0d;
}


pkt_t xdata *adc_report(pkt_t xdata *next) {
  // allocate a packet, take a set of readings.  Note that adc_read()
  // is asynchronous, and may take significant time to complete.
  pkt_t xdata *pkt = pkt_alloc();
  pkt_type(pkt) = SEG_TYPE_ADC;
  pkt_size(pkt) = sizeof(adcPayload);
  adc_read((adcPayload xdata *)pkt_payload(pkt));
  pkt_next(pkt) = next;
  return pkt;
}
```

## File: appr.h

```
#ifndef APPR_H
#define APPR_H
// -*- Mode: C++ -*-
//
// File:        appr.h
// Description: Application Receive process
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "pkt.h"

void appr_recvPkt(pkt_t xdata *pkt);
// stuff a received packet into the receive queue, notify
// the receive process

// void appr_task(void) _task_ APPR_TASK {
// application receive thread.

void appr_didXmit(pkt_t xdata *pkt);
// called from the mac layer immediately after a packet has been
// passed to the radio and just before it is freed.

#endif
```

## File: appr.c

```
// -*- Mode: C++ -*-
//
// File:        appr.c
// Description: Application Receive: manage incoming packets
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "adc.h"
#include "appx.h"
#include "arbor.h"
#include "arq.h"
#include "costTable.h"  // ct_costTo()
#include "grad.h"       // grad_seqno()
#include "id.h"
#include "sync.h"
#include "pkt.h"
#include "mac.h"
#include "screen.h"
#include "stats.h"
#include "vector.h"
#include <rtx51tny.h>   // for os_wait()...
#include <stdio.h>


// queue for received packets waiting for processing by APPR_TASK
```

```
// (the application receive thread)
//
#define APPR_QUEUE_SIZE 10
DEFINE_VECTOR(gAppRQueue, APPR_QUEUE_SIZE);

// ================================================================
// Queue up a received packet.  Packets are put here by the radio
// receive thread and are removed by the application process.  If
// the queue fills up, dump the oldest.
// ## must not be called until APPR_TASK has been started

void appr_recvPkt(pkt_t xdata *pkt) {
  // handle a packet received by the radio process by putting in the
  // APP receive queue and notifying the APR task.  Normally called
  // from within RADR_TASK
  //
  pkt_free(vector_shove(VECTOR(gAppRQueue), pkt));
  // stats_appQueuePkt(vector_count(VECTOR(gAppRQueue)));
  os_send_signal(APPR_TASK);    // notify app task there's a packet
}

// ================================================================
// Application Receive task
//
// Wait for a packet to arrive in the receive queue, then distribute
// the packet to the various services that might want to know about
// it.

static void appr_servicePkt(pkt_t xdata *pkt, pkt_t xdata *gradSeg);

void appr_task(void) _task_ APPR_TASK {

  // one-time initialization of the application's receive queue
  vector_init(VECTOR(gAppRQueue), APPR_QUEUE_SIZE);

  while (1) {
    pkt_t xdata *pkt;
    pkt_t xdata *gradSeg;

    while ((pkt = vector_dequeue(VECTOR(gAppRQueue))) == NULL) {
      os_wait2(K_SIG, 0);       // appr_recvPkt() will generate signal
    }

    gradSeg = grad_find_segment(pkt);
    if (gradSeg == NULL) {
      // no routing header?  Pass it along anyway...
      appr_servicePkt(pkt, gradSeg);
      pkt_free(pkt);

    } else if (!grad_segIsFresh(gradSeg)) {
      // packet is stale - drop now
      SCREEN_TASK(("stale"));
      pkt_free(pkt);

    } else {
      if (grad_isForMe(gradSeg)) {
        appr_servicePkt(pkt, gradSeg);
      }
      // relay or drop the message.  Either way, pkt is guaranteed
      // to be freed by grad_relayIfNeeded().
      grad_relayIfNeeded(pkt, gradSeg);
    }
  }
}

static void appr_servicePkt(pkt_t xdata *pkt, pkt_t xdata *grad) {
  // always print received packet in hex on serial port

#ifdef SCREEN_ENABLE
  screen_goto(14, 1);
#endif
  pkt_dumpHex(pkt);

  // do additional servicing as needed
  while (pkt != NULL) {
    switch (pkt_type(pkt)) {
    case SEG_TYPE_GRADIENT:
    case SEG_TYPE_DISCOVERY:
      // grad updates already happened in appr_task() above
      break;
    case SEG_TYPE_ARQ:
      arq_serviceArq(pkt, grad);
      break;
    case SEG_TYPE_ACK:
      arq_serviceAck(pkt);
      break;
    case SEG_TYPE_APPX:
      appx_serviceSeg(pkt);
      break;
    case SEG_TYPE_PING:
      sync_serviceSeg(pkt);
      break;
    case SEG_TYPE_TEXT:
    case SEG_TYPE_COST_L:
    case SEG_TYPE_COST_H:
    case SEG_TYPE_STATS:
    case SEG_TYPE_ADC:
    case SEG_TYPE_TIME:
    default:
      // contents of the packet has already been printed (above)
      break;
    }
    pkt = pkt_next(pkt);
  }
}

void appr_didXmit(pkt_t xdata *pkt) {
  // Called after pkt has been transmitted by the radio, appr_didXmit()
  // gives individual services a chance to take some action when the
  // transmission has finished.  Notably, arq needs a chance to grab
  // a copy of any packets that need rescheduling
  arq_didXmit(pkt);
}
```

## File: appx.h

```
#ifndef APPX_H
#define APPX_H
// -*- Mode: C++ -*-
//
// File:        appx.h
// Description: Application Transmit support
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "arbor.h"
#include "pkt.h"

// Application Transmit process periodically sends info from this
// node to a designated collection point.

typedef struct _appxPayload {
  node_id fHost;                // host to send data to
  unsigned int fDelayTics;      // inter-report delay (in system tics)
} appxPayload;

void appx_startReporting(node_id destination);
// broadcast a SEG_TYPE_APPX to all nodes, asking them to start
// sending reports to the named destination node.

void appx_stopReporting();
// broadcast a SEG_TYPE_APPX to all nodes, asking them to stop
// sending reports.

pkt_t xdata *appx_makeSeg(pkt_t xdata *next, node_id host);
// allocate a appx segment, link it in with next

void appx_serviceSeg(pkt_t xdata *seg);
// act upon an incoming appx message

#endif
```

## File: appx.c

```
// -*- Mode: C++ -*-
//
// File:        appx.c
// Description: Application Transmit: generate periodic reports
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "adc.h"
#include "appx.h"
#include "arq.h"        // for arq_makeArq()
#include "constell.h"
#include "costTable.h"  // for ct_report(), ct_reset_except()
#include "grad.h"
#include "id.h"
#include "mac.h"
#include "sync.h"
#include "screen.h"
#include "stats.h"
```

```
#include <limits.h>
#include <rtx51tny.h>   // for os_wait()...
#include <stdio.h>      // printf
#include <stdlib.h>     // rand()
#include <string.h>

// sDirectives stores the recipient for reports and the report interval
static appxPayload sDirectives;

// Note that even though there's a slot in an appxPayload packet
// specifically to control the delay time between appx reports, it
// will always be set to DEFAULT_APPX_DELAY.  This could be made
// variable, if needed.
#define DEFAULT_APPX_DELAY 1060

// set a limit to the minimum delay between appx reports
#define MIN_APPX_DELAY (OS_TICS_PER_SECOND/2)

// ================================================================

void appx_startReporting(node_id destination) {
  // broadcast a SEG_TYPE_APPX to all nodes, asking them to start
  // sending reports to the named destination node.
  mac_xmitPkt(grad_makePkt(appx_makeSeg(NULL, destination),
BROADCAST_NODE));
}

void appx_stopReporting() {
  // broadcast a SEG_TYPE_APPX to all nodes, asking them to stop
  // sending reports.
  mac_xmitPkt(grad_makePkt(appx_makeSeg(NULL, 0), BROADCAST_NODE));
}

pkt_t xdata *appx_makeSeg(pkt_t xdata *next, node_id host) {
  // create a SEG_TYPE_APPX packet, requesting that nodes send periodic
  // reports to <host>
  pkt_t xdata *seg = pkt_alloc();
  appxPayload xdata *xp = pkt_payload(seg);
  pkt_type(seg) = SEG_TYPE_APPX;
  pkt_size(seg) = sizeof(appxPayload);
  xp->fHost = host;
  xp->fDelayTics = DEFAULT_APPX_DELAY;
  pkt_next(seg) = next;
  return seg;
}

void appx_serviceSeg(pkt_t xdata *seg) {
  // When a SEG_TYPE_APPX packet is received, copy the appxPayload to
  // the local state and notify the APPX_TASK.  The APPX_TASK will
  // start sending the requested information to the host specified in
  // the header.
  //
  // As a side effect, an APPX packet also resets all the statistics
  // information and routing table info for the node.  Short of creating
  // a new packet type, this is the only convenient way to clear all
  // the logging and statistics info.
  //
  appxPayload xdata *xp = pkt_payload(seg);
  SCREEN_TASK(("appx_ss()")); PKT_PRINT(seg);
  memcpy(&sDirectives, xp, sizeof(appxPayload));
  stats_reset();                // reset packet statistics
  sync_reset();                 // reset sync statistics
  ct_reset_except(xp->fHost);   // reset cost table except to host
  os_send_signal(APPX_TASK);
}

static void _wrapAndSend(pkt_t xdata *pkt) {
  // "Decorate" pkt with a request for reply (arq) and a grad
  // header before passing it to the MAC system for transmission.
  LED_ON(GREEN_LED);
  mac_xmitPkt(grad_makePkt(arq_makeArq(pkt), sDirectives.fHost));
  LED_OFF(GREEN_LED);
}

static void _bide(unsigned int tics) {
  // wait for the given number of tics to elapse.  Each tic is
  // approx 9.6 mSec, or 106 tics per second.
  while (tics != 0) {
    unsigned char t = (tics > UCHAR_MAX)?UCHAR_MAX:tics;
    os_wait2(K_TMO, t);
    tics -= t;
  }
}

// Wait until we're directed to send our status to a particular host,
// then start sending periodic updates.
//
void appx_task(void) _task_ APPX_TASK {
  // one-time initialization
  sDirectives.fHost = 0;
  sDirectives.fDelayTics = 0;

  while (1) {
    SCREEN_TASK(("appx_task(1)"));

    while (sDirectives.fHost == 0) {
      os_wait2(K_SIG, 0);
      // before starting the appx process, choose a random delay
      // to cut down on collisions
      os_wait2(K_TMO, rand());
    }

    // enforce minimum delay
    if (sDirectives.fDelayTics < MIN_APPX_DELAY) {
      sDirectives.fDelayTics = MIN_APPX_DELAY;
    }

    SCREEN_TASK(("h=%bx d=%0bx", sDirectives.fHost,
sDirectives.fDelayTics));

    // I: send low half of cost table and sync state
    _wrapAndSend(ct_report_l(sync_report(NULL)));
    _bide(sDirectives.fDelayTics);

    // II: send high half of cost table and sync state
    _wrapAndSend(ct_report_h(sync_report(NULL)));
```

```
    _bide(sDirectives.fDelayTics);

    // III: send packet statistics and ADC values
    _wrapAndSend(stats_report(adc_report(NULL)));
    _bide(sDirectives.fDelayTics);
  }
}
```

# File: arbor.h

```
#ifndef ARBOR_H
#define ARBOR_H
// -*- Mode: C++ -*-
//
// File:        arbor.h
// Description: General system definitions for ArborNet nodes
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#ifndef NULL
#define NULL (void *)0
#endif

// pervasive data types

typedef unsigned char node_id;
typedef unsigned char cost_t;
typedef unsigned char seq_t;

// too handy not to define

#define BITMASK(b7,b6,b5,b4,b3,b2,b1,b0) \
    ((b7<<7)|(b6<<6)|(b5<<5)|(b4<<4)|(b3<<3)|(b2<<2)|(b1<<1)|(b0))

// units for os_wait(), assuming 12Mhz clock
#define OS_TICS_PER_SECOND      106

// system-wide definitions
```

```
//
#define MAIN_TASK       0
#define RADR_TASK       1       // radio receive thread
#define MAC_TASK        2       // radio transmit thread
#define SYNC_TASK       3       // periodic ping thread
#define APPR_TASK       4       // packet receiver task
#define ARQ_TASK        5       // retry packets until ack'd
#define APPX_TASK       6       // send data to collection point

#endif // ARBOR_H
```

# File: arbor.c

```
// -*- Mode: C++ -*-
//
// File:        arbor.c
// Description: initialization and main process loop for ArborNet
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// Main startup file for arbor system.

#include "arbor.h"

#include "adc.h"        // adc_init()
#include "appx.h"       // appx_startReporting()
#include "arq.h"        // arq_init()
#include "constell.h"   // for LEDs
#include "costTable.h"  // ct_init()
#include "grad.h"       // grad_init()
#include "id.h"         // nodeName()...
#include "pkt.h"        // pkt_init()
#include "rad.h"        // rad_init()
#include "screen.h"
#include "serial.h"     // serial_init()
#include "stats.h"      // stats_reset()
#include "sync.h"       // sync_getLocalTime()
#include <aduc824.h>    // for PLLCON
#include <rtx51tny.h>   // for os_wait()...
#include <stdio.h>      // puts()
#include <stdlib.h>     // srand()

#define BIDE(tics) os_wait2(K_TMO, (tics))
// sleep for the given number of tics.  Each tic is approximately
// 9.4 mSec.  tics must be less than 256.

void test_leds() {
  int i;

  for (i=0; i<2; i++) {
    LED_ON(AMBER_LED);
    LED_OFF(YELLOW_LED);
    BIDE(20);
    LED_ON(RED_LED);
    LED_OFF(AMBER_LED);
    BIDE(20);
    LED_ON(ORANGE_LED);
    LED_OFF(RED_LED);
    BIDE(20);
    LED_ON(GREEN_LED);
    LED_OFF(ORANGE_LED);
    BIDE(20);
    LED_ON(YELLOW_LED);
    LED_OFF(GREEN_LED);
    BIDE(20);
  }
  LED_OFF(YELLOW_LED);
}

// Install jumper_0 to blast packets directly to the radio.
// Used for debugging.
static void _blast_packets() {
  os_delete_task(SYNC_TASK);
  while (JUMPER_0()) {
    // blast packets
    pkt_t xdata *pkt = adc_report(NULL);
    rad_xmitPkt(pkt);
    pkt_free(pkt);
    os_wait2(K_TMO, 20);
  }
  os_create_task(SYNC_TASK);    // restart sync task.
}

// ===================================================================
// ===================================================================
// program entry point here
```

```
void init() _task_ MAIN_TASK {
  // experiment to see if setting radio pins early makes a difference...
  TR1000_CTL_DIRECTION = BITMASK(0,0,0,0,0,1,1,0); // setup mode ctl pins
  TR1000_CTL_DRIVE = 0x00;      // standard CMOS I/O
  TR1000_CTL0 = 1;              // set TR1000 to receive mode
  TR1000_CTL1 = 1;

  BIDE(4);
  PLLCON = 0x00; // 12 MHz

    TIMECON = 0x13;     // configure TIMECON to:
                        // x0xxxxxx - count hours 0 to 255
                        // xx01xxxx - count in seconds
                        // xxxx0xxx - auto reload TIC
                        // xxxxx0xx - clear TIC interrupt flag
                        // xxxxxx1x - enable counting of TIC
                        // xxxxxxx1 - enable counting of RTC

  LED_INIT();
  test_leds();
  srand(nodeID());

  serial_init();                // set up baud rate
  pkt_init();                   // init packet storage
  BIDE(10);
  printf("\r\n\r\narbor x0.10 %s\r\n", nodeName());

  LED_ON(AMBER_LED);

  adc_init();                   // initialize analog module
  ct_init();                    // init cost tables
  grad_init();                  // init gradient routing module
  rad_init();                   // init radio module
  stats_reset();

  LED_OFF(AMBER_LED);

  os_create_task(MAC_TASK);     // start mac process
  os_create_task(APPR_TASK);    // app receive thread
  os_create_task(APPX_TASK);
  os_create_task(ARQ_TASK);
  // MAC_TASK must be started before SYNC_TASK
  os_create_task(SYNC_TASK);    // start sync task.
  os_create_task(RADR_TASK);    // restart receiver

  SCREEN_CLEAR();

  while (1) {
    unsigned char prevRTC, tmp;

    SCREEN_TASK(("%s command: ", nodeName()));

    // This loop does two things: It breaks when a character has been
    // typed on the serial input.  While it's waiting, it flashes the
    // yellow LED whenever the local clock crosses a 2 second boundary.
    // This should give a visual indication of synchronization among
    // nodes.
    //
    // Note that sync_getLocalTime() returns time in units of 128ths
    // of a second.  When assigned to prevRTC and tmp, it's truncated
    // to an unsigned char, or 256 tics, or two seconds.
    prevRTC = sync_getLocalTime();      // truncated to 2 seconds
    while (!serial_charIsAvailable()) {

      // blast packets if jumper 0 installed
      if (JUMPER_0()) _blast_packets();

      tmp = sync_getLocalTime();        // also truncated...
      if (tmp < prevRTC) {              // virtual RTC rolled over
        LED_ON(YELLOW_LED);
        if ((sync_getLocalTime() & 0xff00) == 0) {
          os_wait2(K_TMO, 20);  // long flash on the minute
        } else {
          os_wait2(K_TMO, 1);   // blip otherwise
        }
        LED_OFF(YELLOW_LED);
      } else {
        // One RTC tic is 7.8125 mSec.  One OS tic is 9.44 tics.
        // We expect tmp to roll over in 256 - tmp RTC tics, so
        // we conservatively wait (256-tmp)/2 OS tics before
        // checking again.
        os_wait2(K_TMO, (256-tmp)>>1);
        // os_wait2(K_TMO, ((256-tmp)>>1)+10);
      }
      prevRTC = tmp;
    }

    // here when a serial char became available
    tmp = getchar();
    if (tmp == 'H') {
      printf("...start reporting");
      appx_startReporting(nodeID());
    } else if (tmp == 'S') {
      printf("...stop reporting");
      appx_stopReporting();
    } else {
      putchar('?');
    }
  }
}
```

# File: arq.h

```
#ifndef ARQ_H
#define ARQ_H
// -*- Mode: C++ -*-
//
// File:        arq.h
// Description: Header file for Automatic Reply reQuest mechanism
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "pkt.h"

// an originator that wants a reply installs an arq segment in
// the message.  The receiver will send an ack packet in reply.

// segments with SEG_TYPE_ARQ or SEG_TYPE_ACK have this
// as their payload.  The originator and destination
// ids are assumed to be available in a grad seg in the
// same packet.  the fTimeout and fRetries fields are
// artifacts that simplify the programming and offer
// a little debugging info.

// BIG OL' NOTE: The ARQ_DETAULT_RETRIES has been set to zero, which
// effectively prevents the ARQ system from ever sending repeat packets.
// The number of ARQ (requests) issued and the number of ACK (replies)
// are logged in the statistics, though.
//
// This is because the ARQ/ACK sytem has been shown to work, but for
// testing, we don't want the variability introduced by repeated ARQ
// packets -- we'd rather just drop the packet than retry.

#define ARQ_DEFAULT_RETRIES 0
// ARQ quits re-sending a packet after RETRIES attempts

typedef struct _arqPayload {
  unsigned char fReference;     // packet ID
  unsigned char fRetries;       // # of times remaining
} arqPayload;

typedef struct _ackPayload {
  unsigned char fReference;
} ackPayload;

pkt_t xdata *arq_makeArq(pkt_t xdata *pkt);
// install SEG_TYPE_ARQ segment in pkt

pkt_t xdata *arq_makeAck(pkt_t xdata *pkt, unsigned char reference);
// install SEG_TYPE_ACK segment in pkt

void arq_serviceArq(pkt_t xdata *seg, pkt_t xdata *grad);
// Handle an incoming ARQ packet.  Respond by creating an ACK packet and
// sending it to the originator.

void arq_serviceAck(pkt_t xdata *seg);
// Handle an incoming ACK packet.  Respond by finding and deleting the
// corresponding packet in the retry queue.

void arq_didXmit(pkt_t xdata *pkt);
// Called by the MAC thread when a packet is sent.  If the packet
// contains an ARQ header and its retry count is greater than 0,
// create a copy of the packet and install the copy in the retry
// queue.

// void arq_task(void) _task_ ARQ_TASK
// Task regularly examines retry queue.  If there is a packet in the
// retry queue, remove it from the queue and pass it to the MAC layer
// for transmission.

#endif
```

# File: arq.c

```
// -*- Mode: C++ -*-
//
// File:        arq.c
// Description: Automatic Reply reQuest: manage ARQ and ACK packets
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
```

```
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "appr.h"
#include "arbor.h"
#include "arq.h"
#include "grad.h"
#include "id.h"
#include "mac.h"
#include "pkt.h"
#include "screen.h"
#include "stats.h"
#include "vector.h"
#include <rtx51tny.h>   // for os_wait()...
#include <stdio.h>

// The gRetryQueue is the home for packets awaiting an ACK
// from a remote host.
#define RETRY_QUEUE_SIZE 4
DEFINE_VECTOR(gRetryQueue, RETRY_QUEUE_SIZE);

#define ARQ_INTERVAL_TICS OS_TICS_PER_SECOND
// Once every ARQ_INTERVAL_TICS, examine the retry queue.  If there
// is a packet available, remove it and pass it to the MAC layer for
// retransmission.  This limits the maximum rate of retries.

static unsigned char gReference;
// A reference number for each ACK packet generated.

// ================================================================
// ================================================================
// public routines

pkt_t xdata *arq_makeArq(pkt_t xdata *pkt) {
  // add an arq segment to this packet
  pkt_t xdata *arqSeg = pkt_alloc();
  arqPayload xdata *ap = pkt_payload(arqSeg);

  ap->fReference = gReference++;
  ap->fRetries = ARQ_DEFAULT_RETRIES;
  pkt_size(arqSeg) = sizeof(arqPayload);
  pkt_type(arqSeg) = SEG_TYPE_ARQ;
  pkt_next(arqSeg) = pkt;       // link to main packet

  return arqSeg;
}

pkt_t xdata *arq_makeAck(pkt_t xdata *pkt, unsigned char reference) {
  // add an ack segment to packet
  pkt_t xdata *ackSeg = pkt_alloc();
  ackPayload xdata *ap = pkt_payload(ackSeg);

  ap->fReference = reference;
  pkt_size(ackSeg) = sizeof(ackPayload);
  pkt_type(ackSeg) = SEG_TYPE_ACK;
  pkt_next(ackSeg) = pkt;       // link to main packet

  stats_arq();                  // log a request for acknowlegement

  return ackSeg;
}

void arq_serviceArq(pkt_t xdata *seg, pkt_t xdata *grad) {
  // Handle an incoming ARQ packet.  Respond by creating an ACK packet
  // and sending it to the originator.  seg is known to be a segment of
  // type SEG_TYPE_ARQ, grad (if non null) is SEG_TYPE_GRAD.
  gradPayload xdata *gp;
  arqPayload xdata *ap;

  // can't handle a packet with no return address
  if (grad == NULL) return;
  gp = (gradPayload xdata *)pkt_payload(grad);
  ap = (arqPayload xdata *)pkt_payload(seg);

  mac_xmitPkt(grad_makePkt(arq_makeAck(NULL, ap->fReference),gp-
>fOriginator));
}


void arq_serviceAck(pkt_t xdata *seg) {
  // seg is known to be SEG_TYPE_ACK, and part of a packet targeted
  // for this node.  If it is an acknowlegement for an ARQ packet
  // festering in the retry queue, now is the time to delete it.

  // ## don't call before ARQ_TASK is started

  ackPayload xdata *ackh = pkt_payload(seg);
  unsigned char i = VECTOR(gRetryQueue)->fCount;

  stats_ack();          // log an acknowlegement received

  while (i--) {
    pkt_t xdata *retryPkt;
    pkt_t xdata *arqSeg;
    retryPkt = fast_vector_ref(VECTOR(gRetryQueue), i);
    arqSeg = pkt_find_segment(retryPkt, SEG_TYPE_ARQ);
    if (arqSeg != NULL) {
      arqPayload xdata *arqh = pkt_payload(arqSeg);
```

```
      if (arqh->fReference == ackh->fReference) {
        // found a match.  Remove retryPkt from the retry queue.
        pkt_free(fast_vector_remove(VECTOR(gRetryQueue), i));
        return;
      }
    }
  }
}

void arq_didXmit(pkt_t xdata *pkt) {
  // Called by the MAC thread when a packet is sent.  If this node
  // is the originator and it has an ARQ header and its retry count
  // is greater than 0, create a copy of the packet and install it
  // in the retry queue.

  // ## don't call before ARQ_TASK is started

  pkt_t xdata *arqSeg;
  pkt_t xdata *gradSeg;
  arqPayload xdata *ap;

  if ((arqSeg = pkt_find_segment(pkt, SEG_TYPE_ARQ)) == NULL) {
    // no ARQ segment in the packet - fuggeddaboudit
    return;
  }

  if (((gradSeg = grad_find_segment(pkt)) == NULL) ||
      (((gradPayload xdata *)pkt_payload(gradSeg))->fOriginator !=
nodeID())) {
    // we weren't the originator
    return;
  }

  ap = (arqPayload xdata *)pkt_payload(arqSeg);
  if (ap->fRetries-- > 0) {
    // There are one or more retries left in this packet.  Make a
    // copy of the packet and install the copy in the retry queue.
    pkt = pkt_copy(pkt);        // make a copy
    pkt_free(vector_shove(VECTOR(gRetryQueue), pkt));
    os_send_signal(ARQ_TASK);   // tell ARQ_TASK to check retry queue
  }
}

void arq_task(void) _task_ ARQ_TASK {
  // Task regularly examines the retry queue, passing messages to the
  // MAC layer for retransmission as they become available.
  pkt_t xdata *pkt;

  // one time initialization
  vector_init(VECTOR(gRetryQueue), RETRY_QUEUE_SIZE);
  gReference = 0;

  while (1) {

    do {
      // Block until there might be a packet in the retry queue.
      if (vector_count(VECTOR(gRetryQueue)) == 0) {
        os_wait2(K_SIG, 0);     // arq_didXmit() will generate signal
      }
      // This wait() regulates the max rate at which retries are sent
      os_wait2(K_TMO, ARQ_INTERVAL_TICS);
      // the packet may have been removed by serviceAck() while we were
      // waiting.  Check if the packet is still there before continuing.
    } while ((pkt = vector_dequeue(VECTOR(gRetryQueue))) == NULL);

    // pkt is the packet to be retransmitted.  Update grad info
    // and pass ot MAC layer for retransmission
    grad_updateSeg(pkt);
    mac_xmitPkt(pkt);
    SCREEN_TASK(("arq_task(3) %bu", vector_count(VECTOR(gRetryQueue))));
  }
}
```

# File: constell.h

```
#ifndef _CONSTELL_H_
#define _CONSTELL_H_
// -*- Mode: C++ -*-
//
// File:        constell.h
// Description: hardware-specific definitions for Constellation board
// Author:      Paul Pham
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#define PSD_REG_BASE   0x2000

// general structure of 8-bit register allowing bit access
typedef struct
{
    unsigned char bit0 : 1;
    unsigned char bit1 : 1;
    unsigned char bit2 : 1;
    unsigned char bit3 : 1;
    unsigned char bit4 : 1;
    unsigned char bit5 : 1;
    unsigned char bit6 : 1;
    unsigned char bit7 : 1;
} Register;

// union allowing either byte or bit access to 8-bit register
typedef union
{
   char byte;
   Register bits;
} Mixed_Reg;

// address offsets of PSD control registers
#define DATAIN_A          ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x00))
#define DATAIN_B          ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x01))
#define DATAIN_C          ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x10))
#define DATAIN_D          ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x11))
#define DATAOUT_A         ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x04))
#define DATAOUT_B         ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x05))
#define DATAOUT_C         ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x12))
#define DATAOUT_D         ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x13))
#define DIRECTION_A       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x06))
#define DIRECTION_B       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x07))
#define DIRECTION_C       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x14))
#define DIRECTION_D       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x15))
#define DRIVE_A           ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x08))
#define DRIVE_B           ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x09))
#define DRIVE_C           ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x16))
#define DRIVE_D           ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x17))
#define OUTENABLE_A       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x0C))
#define OUTENABLE_B       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x0D))
#define OUTENABLE_C       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x1A))
#define OUTENABLE_D       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x1B))
#define CONTROL_A         ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x02))
#define CONTROL_B         ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x03))
#define IMC_A             ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x0A))
#define IMC_B             ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x0B))
#define IMC_C             ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x18))
#define OMC_AB            ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x20))
#define OMC_BC            ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x21))
#define OMCMASK_AB        ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x22))
#define OMCMASK_BC        ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0x23))
#define MAINPROTECT       ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0xC0))
#define ALTPROTECT        ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0xC2))
#define JTAG              ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0xC7))
#define PMMR0             ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0xB0))
#define PMMR2             ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0xB4))
#define PAGE              ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0xE0))
#define VM                ((volatile Mixed_Reg xdata *)(PSD_REG_BASE+0xE2))

//PSD PORTA
#define PA0               bit0
#define PA1               bit1
#define PA2               bit2
#define PA3               bit3
#define PA4               bit4
#define PA5               bit5
#define PA6               bit6
#define PA7               bit7

//PSD PORTB
#define PB0               bit0
#define PB1               bit1
#define PB2               bit2
#define PB3               bit3
```

```
#define PB4               bit4
#define PB5               bit5
#define PB6               bit6
#define PB7               bit7

//PSD PORTC
#define PC0               bit0
#define PC1               bit1
#define PC2               bit2
#define PC3               bit3
#define PC4               bit4
#define PC5               bit5
#define PC6               bit6
#define PC7               bit7

//PSD PORTD
#define PD0               bit0
#define PD1               bit1
#define PD2               bit2

//PSD JTAG
#define JEN               bit0   // JTAG enable

//PSD PMMR0
#define APD_ENABLE        bit1
#define PLD_TURBO         bit3
#define PLD_ARRAY_CLK     bit4
#define PLD_MCELL_CLK     bit5

//PSD PMMR2
#define PLD_CNTL0         bit2
#define PLD_CNTL1         bit3
#define PLD_CNTL2         bit4
#define PLD_ALE           bit5
#define PLD_DBE           bit6

//PSD VM
#define SRAM_CODE         bit0
#define EE_CODE           bit1
#define FL_CODE           bit2
#define EE_DATA           bit3
#define FL_DATA           bit4
#define PIO_EN            bit7

// Flash parameters

#define NVM_DATA_POLL     0x80  // flash status "data poll" bit at DQ7
#define NVM_DATA_TOGGLE   0x40  // flash status "toggle poll" bit at DQ6
#define NVM_ERROR         0x20  // flash status "error" bit at DQ5

// For F2 with EEPROM boot
#define MAX_EEPROM_RETRY  0x0FFF
// Maximum number of attemps to check status after
// a write opertaion to EEPROM

// sfr PLLCON = 0xD7;

#define TR1000_CTL0           DATAOUT_D->bits.PD1
#define TR1000_CTL1                       DATAOUT_D->bits.PD2
#define TR1000_CTL_DIRECTION  DIRECTION_D->byte
#define TR1000_CTL_DRIVE      DRIVE_D->byte
#define BART_DATA_DIRECTION DIRECTION_B->byte
#define BART_DATA_DRIVE     DRIVE_B->byte
#define BART_DATA_CONTROL   CONTROL_B->byte
#define BART_DATA_OUT       DATAOUT_B->byte
#define BART_DATA_IN        DATAIN_B->byte
#define LED_DIRECTION     DIRECTION_A->byte
#define LED_DRIVE         DRIVE_A->byte
#define LED_CONTROL       CONTROL_A->byte
#define LED_SET           DATAOUT_A
#define BART_READY        INT0
#define BART_CLOCK        T0
#define BART_MODE         T1

// values for BART_MODE
#define BART_MODE_XMIT 0
#define BART_MODE_RECV 1

// values for xxx_DIRECTION registers
#define XMIT 0xFF
#define RECV 0x00

#define LED_INIT()  LED_DIRECTION = 0x1F; \
                                LED_DRIVE     = 0x00; \
                                LED_CONTROL   = 0x00; \
                    LED_SET->byte = 0xFF

// To turn LED on, pull pin down to ground because other end is connected to
// Vcc. To turn LED off, pull pin up.

#define LED_ON(b)       LED_SET->bits.b = 0
#define LED_OFF(b)      LED_SET->bits.b = 1
#define ALL_LEDS_ON()   LED_SET->byte = 0x00
#define ALL_LEDS_OFF()  LED_SET->byte = 0xff

#define AMBER_LED   PA0
#define RED_LED     PA1
#define ORANGE_LED  PA2
#define GREEN_LED   PA3
#define YELLOW_LED  PA4

// return true when jumper installed
#define JUMPER_0()      (!DATAIN_C->bits.PC2)
#define JUMPER_1()      (!DATAIN_C->bits.PC7)

#endif  // CONSTELL_H
```

## File: costTable.h

```
// -*- Mode: C++ -*-
//
// File:        costTable.h
// Description: header file for cost table routines
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#ifndef COST_TABLE_H
#define COST_TABLE_H

#include "arbor.h"
#include "grad.h"   // for gradPayload
#include "pkt.h"        // for pkt_t def

#define COST_UNKNOWN ((cost_t)0xff)

// A costRecord represents this node's cost to a given originator.  A
// message leaves behind a trial of costRecords as it hops from node
// to node.

// ## NB: Since the cost table is now layed out with the N'th entry
// corresponding to fOriginator == N, the fOriginator field in the
// costRecord structure isn't really required.

typedef struct {
  node_id fOriginator;          // originator of this costRecord
  seq_t   fSequence;            // orignator's sequence number
  cost_t  fCost;                // accrued cost since origination
} costRecord;

// Support for costRecords

void ct_init();
// initialize the cost table

void ct_reset_except(node_id target);
// reset all entries in the cost table except for
// target.  If target is out of range, resets all.

bit ct_update(gradPayload xdata *gh);
// Create or update a costRecord for originator, returning true if
// originator/sequence pair corresponds to fresh message.  Even if the
// message is stale, the hops field is updated if the new record is
// advantageous.

bit ct_shouldRelay(gradPayload xdata *gh, bit isDiscovery);
// Look up the cost for destination in the routing table.  If a record
// for the destination doesn't exist, return false.  If a record does
// exist, return true if the cost table's hop count is smaller than
// the given budget.  Otherwise return false.

cost_t ct_costTo(node_id node);
// Return the cost to the given node, or COST_UNKNOWN if there is no
// costRecord in the routing table.

pkt_t xdata *ct_report_l(pkt_t xdata *next);
pkt_t xdata *ct_report_h(pkt_t xdata *next);
// return packets with low half and high half of routing table.

#endif
```

## File: costTable.c

```
// -*- Mode: C++ -*-
//
// File:        costTable.c
// Description: maintain cost estimates to other nodes
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
```

```
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

//
// Support for cost tables
//
// 08Nov2000 r@media.mit.edu

// Modified fancy LRU cost table to direcly indexed one node/one entry
// form.  This is simple and fast, but not scalable.

#include "costTable.h"
#include "id.h"
#include <stdio.h>

// ================================================================
// the actual storage for vector and costRecords

#define MAX_COSTRECORDS         26
#define COST_REPORT_SIZE        13

// IDs are biased by this offset
#define MIN_ID 'A'

// allocate a static pool of cost records
static costRecord xdata _costRecords[MAX_COSTRECORDS];

// ================================================================
// internal procedures

static int _findRecord(node_id originator);
static void _createRecord(gradPayload xdata *gp);

#define NULL_ORIGINATOR (node_id)0

static int _recordIndex(node_id originator) {
  // map a node id to a cost table index, returning -1 if the
  // node id is out of range.
  if ((originator < MIN_ID) || (originator >= (MIN_ID + MAX_COSTRECORDS))) {
    return -1;
  } else {
    return originator-MIN_ID;
  }
}

static void _createRecord(gradPayload xdata *gp) {
  // copy gradPayload's salient points into the cost table.
  unsigned char index = _recordIndex(gp->fOriginator);
  if (index != -1) {
    costRecord xdata *cr = &_costRecords[index];
    cr->fOriginator = gp->fOriginator;
    cr->fSequence = gp->fSequence;
    cr->fCost = gp->fCostAccrued;
  }
}

// ================================================================
// exported procedures

void ct_init() {
  // initialize the cost table with unused cost records
  ct_reset_except(NULL_ORIGINATOR);
}

void ct_reset_except(node_id target) {
  // clear cost entry for all nodes except the specified target
  unsigned char i;

  for (i=0; i<MAX_COSTRECORDS; i++) {
    costRecord xdata *cr;
    if (i != _recordIndex(target)) {
      cr = &_costRecords[i];
      cr->fOriginator = NULL_ORIGINATOR;
      cr->fCost = COST_UNKNOWN;
    }
  }
}

#define IS_NEWER(seqA, seqB) ((((seqB)-(seqA)) & 0xf0) != 0)
// IS_NEWER() returns true if sequence number A is newer than sequence
// number B, in this case, within 16 counts.  A and B are interpreted to
// be MOD 256 (one byte).

bit ct_update(gradPayload xdata *gp) {
  // Create or update a cost record for originator, returning true if
  // originator/sequence pair corresponds to fresh message.  Even if
  // the message is stale, the cost field is updated if the new record
  // is advantageous.
  unsigned char index;
  costRecord xdata *cr;

  if (gp->fOriginator == nodeID()) {
    // I was the originator of this message.  Feggadaboudit.
    return 0;
  }

  index = _recordIndex(gp->fOriginator);

  if (index == -1) {
    _createRecord(gp);
    return 1;
  }

  cr = &_costRecords[index];

  if (IS_NEWER(gp->fSequence, cr->fSequence)) {
```

```
    cr->fSequence = gp->fSequence;
    cr->fCost = gp->fCostAccrued;
    return 1;
  }

  if ((gp->fSequence == cr->fSequence) && (gp->fCostAccrued < cr->fCost)) {
    // this sequence number already seen, but adverized cost is better
    // Update cost estimate
    cr->fCost = gp->fCostAccrued;
  }
  return 0;
}

bit ct_shouldRelay(gradPayload xdata *gp, bit isDiscovery) {
  // if not discovery: look up cost to destination in cost table.
  //    if unknown, return false.
  //    else return true if the message's cost budget is larger than
  //    this node's cost to the destination.
  // if discovery:
  //    return true if the message's cost budget is larger than 0
  cost_t myCost = (cost_t)((isDiscovery)?0:ct_costTo(gp->fDestination));
  if (myCost == COST_UNKNOWN) {
    return 0;
  } else {
    return (gp->fCostBudget > myCost);
  }
}

cost_t ct_costTo(node_id node) {
  // Return the cost to the given node, or COST_UNKNOWN if there is no
  // record in the routing table.
  int i = _recordIndex(node);
  if (i == -1) {
    return COST_UNKNOWN;
  } else {
    costRecord xdata *cr = &_costRecords[i];
    return cr->fCost;
  }
}

static pkt_t xdata *ct_report(seg_type t,
                              unsigned char offset,
                              pkt_t xdata *next) {
  unsigned char i;
  pkt_t xdata *pkt = pkt_alloc();
  unsigned char *p = pkt_payload(pkt);
  pkt_type(pkt) = t;
  pkt_size(pkt) = COST_REPORT_SIZE;
  for (i=0; i<COST_REPORT_SIZE; i++) {
    *p++ = (_costRecords[i+offset]).fCost;
  }
  pkt_next(pkt) = next;
  return pkt;
}

pkt_t xdata *ct_report_l(pkt_t xdata *next) {
  // create a packet with the low half of the routing table in
  // the payload
  return ct_report(SEG_TYPE_COST_L, 0, next);
}

pkt_t xdata *ct_report_h(pkt_t xdata *next) {
  // create a packet with the high half of the routing table in
  // the payload
  return ct_report(SEG_TYPE_COST_H, COST_REPORT_SIZE, next);
}
```

# File: grad.h

```
#ifndef GRAD_H
#define GRAD_H
// -*- Mode: C++ -*-
//
// File:        grad.h
// Description: header file for gradient routing support
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.


#include "arbor.h"
#include "pkt.h"        // for pkt_t

// A gradPayload contains the information used by the gradient routing
// mechanism.  A gradPayload can be found in the payload of any packet
// segment whose type is SEG_TYPE_GRADIENT or SEG_TYPE_DISCOVERY.

typedef struct _gradPayload {
  node_id fOriginator;
  unsigned char fSequence;
  unsigned char fCostAccrued;
  node_id fDestination;
  unsigned char fCostBudget;
} gradPayload;

// a node that will never be a target
#define BROADCAST_NODE  0xff

#define DEFAULT_DISCOVERY_COST  25

// ASSERT(sizeof(gradPayload)<MAX_PAYLOAD_SIZE, "gradPkt payload too big");

void grad_init();
// intialize the gradient routing system

pkt_t xdata *grad_find_segment(pkt_t xdata *pkt);
// find a SEG_TYPE_GRADIENT or SEG_TYPE_DISCOVERY segment in the packet

pkt_t xdata *grad_makePkt(pkt_t xdata *pkt, node_id dest);
// install a GRAd segment in this packet.

void grad_updateSeg(pkt_t xdata *pkt);
// find the SEG_TYPE_DISCOVERY or SEG_TYPE_GRADIENT segment in this
// packet.  In-place, fixup current sequence number and cost info.

bit grad_segIsFresh(pkt_t xdata *gradSeg);
  // seg must refer to a packet segment of type SEG_TYPE_DISCOVERY
  // or SEG_TYPE_GRADIENT.  Updates cost tables, returns true if
  // this packet hasn't been seen before.

bit grad_isForMe(pkt_t xdata *gradSeg);
  // seg must refer to a packet segment of type SEG_TYPE_DISCOVERY
  // or SEG_TYPE_GRADIENT.  Returns true if the packet is destined
  // for this node (dest is either BROADCAST_NODE or this node).

void grad_relayIfNeeded(pkt_t xdata *pkt, pkt_t xdata *gradSeg);
  // seg must refer to a packet segment of type SEG_TYPE_DISCOVERY
  // or SEG_TYPE_GRADIENT.  Relay pkt if appropriate, else just
  // free it.

#endif
```

# File: grad.c

```
// -*- Mode: C++ -*-
//
// File:        grad.c
// Description: To relay or not to relay?  Answered here...
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
```

```
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// GRAd is responsible for the routing of packets.  Upon reception,
// GRAd decides whether to relay a packet, pass it to the application
// level, or to drop it.  For transmission, GRAd appends a routing
// header that will be used by the GRAd service in receiving nodes.

#include "arbor.h"
#include "grad.h"
#include "costTable.h"      // ct_costTo() ...
#include "id.h"             // nodeID()
#include "stats.h"          // stats_origPkt()
#include "mac.h"            // mac_xmitPkt()...
#include "arq.h"            // arq_recvPkt()
#include <stdio.h>

static unsigned char gSequence;
// Sequence number for next originated packet.

// ============================================================
// exported routines

void grad_init() {
  gSequence = 0;
}

pkt_t xdata *grad_find_segment(pkt_t xdata *pkt) {
  // search the list of segments for a packet that contains a
  // gradPayload.
  while (pkt != NULL) {
    seg_type t = pkt_type(pkt);
    if ((t == SEG_TYPE_GRADIENT) || (t == SEG_TYPE_DISCOVERY)) return pkt;
    pkt = pkt_next(pkt);
  }
  return NULL;
}

static seg_type grad_prepPayload(gradPayload xdata *payload, node_id dest)
{
  // fill in a gradPayload with dest, cost, sequence, etc.  Returns
  // SEG_TYPE_DISCOVERY or SEG_TYPE_GRADIENT as appropriate.
  cost_t cost = ct_costTo(dest);
  payload->fOriginator = nodeID();
  payload->fSequence = gSequence++;
  payload->fCostAccrued = 1;        // receivers are already one hop away
  payload->fDestination = dest;
  if (cost == COST_UNKNOWN) {
    payload->fCostBudget = DEFAULT_DISCOVERY_COST;
    stats_floodPkt();           // note a flood packet
    return SEG_TYPE_DISCOVERY;
  } else {
    payload->fCostBudget = cost;
    stats_origPkt();            // note a routed packet
    return SEG_TYPE_GRADIENT;
  }
}

void grad_updateSeg(pkt_t xdata *pkt) {
  // find the SEG_TYPE_DISCOVERY or SEG_TYPE_GRADIENT segment in this
  // packet.  In-place, fixup current sequence number and cost info.
  // Currently, this routine is used to update ARQ packets when they
  // are retransmitted.
  pkt_t xdata *gradSeg;
  gradPayload xdata *gp;

  if ((gradSeg = grad_find_segment(pkt)) == NULL) {
    return;
  }
  gp = (gradPayload xdata *)pkt_payload(gradSeg);
  pkt_type(gradSeg) = grad_prepPayload(gp, gp->fDestination);
}


pkt_t xdata *grad_makePkt(pkt_t xdata *pkt, node_id dest) {
  // Tack a GRAd routing header on the front of this packet.  If the
  // cost to dest is known, it creates a regular GRAd data packet.  If
  // the cost isn't known, it creates a discovery packet.
  pkt_t xdata *seg = pkt_alloc();
  pkt_size(seg) = sizeof(gradPayload);
  pkt_type(seg) = grad_prepPayload((gradPayload xdata *)pkt_payload(seg),
dest);
  pkt_next(seg) = pkt;            // link pkt in as next in line
  return seg;
}

bit grad_segIsFresh(pkt_t xdata *gradSeg) {
  // seg must refer to a packet segment of type SEG_TYPE_DISCOVERY
  // or SEG_TYPE_GRADIENT.  Updates cost tables, returns true if
  // this packet hasn't been seen before.
  return ct_update((gradPayload xdata *)pkt_payload(gradSeg));
}

bit grad_isForMe(pkt_t xdata *gradSeg) {
  // seg must refer to a packet segment of type SEG_TYPE_DISCOVERY
  // or SEG_TYPE_GRADIENT.  Returns true if the packet is destined
  // for this node (dest is either BROADCAST_NODE or this node).
  gradPayload xdata *gp = pkt_payload(gradSeg);
  return (gp->fDestination == nodeID()) ||
         (gp->fDestination == BROADCAST_NODE);
}

void grad_relayIfNeeded(pkt_t xdata *pkt, pkt_t xdata *gradSeg) {
  // seg must refer to a packet segment of type SEG_TYPE_DISCOVERY
  // or SEG_TYPE_GRADIENT.  Relay pkt if appropriate, else just
```

```
  // free it.
  gradPayload xdata *gp = pkt_payload(gradSeg);
  if (ct_shouldRelay(gp, pkt_type(gradSeg)==SEG_TYPE_DISCOVERY)) {
    // pkt should be relayed.  Update the grad header and schedule the
    // packet for re-transmission.
    gp->fCostAccrued++;         // one hop further from originator
    gp->fCostBudget--;          // one hop closer to destination
    stats_relayPkt();           // note a relayed packet
    mac_xmitPkt(pkt);
  } else {
    // Packet is not to be relayed.  Drop it.
    pkt_free(pkt);
  }
}
```

# File: id.h

```
#ifndef ID_H
#define ID_H
// -*- Mode: C++ -*-
//
// File:        id.h
// Description: header file for node IDs
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.


char *nodeName();
unsigned char nodeID();

#endif
```

# File: id.c

```
// -*- Mode: C++ -*-
//
// File:        id.c
// Description: defines nodeName() and nodeID()
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
```

```
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "id.h"

// NODE_NAME can be defined from the command line
// Maybe.

#ifndef NODE_NAME
// #define NODE_NAME "Aspen"    // 0x41
// #define NODE_NAME "Beech"    // 0x42
// #define NODE_NAME "Chestnut" // 0x43
// #define NODE_NAME "Dogwood"  // 0x44
// #define NODE_NAME "Elm"      // 0x45
// #define NODE_NAME "Fig"      // 0x46
// #define NODE_NAME "Ginkgo"   // 0x47
// #define NODE_NAME "Holly"    // 0x48
// #define NODE_NAME "Ironwood" // 0x49
// #define NODE_NAME "Juniper"  // 0x4a
// #define NODE_NAME "Kapok"    // 0x4b
// #define NODE_NAME "Linden"   // 0x4c
// #define NODE_NAME "Magnolia" // 0x4d
// #define NODE_NAME "Nyssa"    // 0x4e
// #define NODE_NAME "Olive"    // 0x4f
// #define NODE_NAME "Pear"     // 0x50
// #define NODE_NAME "Quince"   // 0x51
// #define NODE_NAME "Redwood"  // 0x52
// #define NODE_NAME "Sycamore" // 0x53
// #define NODE_NAME "Tupelo"   // 0x54
// #define NODE_NAME "Uri"      // 0x55
// #define NODE_NAME "Viburnum" // 0x56
// #define NODE_NAME "Willow"   // 0x57
// #define NODE_NAME "Xylosma"  // 0x58
// #define NODE_NAME "Yew"      // 0x59
#define NODE_NAME "Zelkova"     // 0x5a
#endif

char *nodeName() {
  return NODE_NAME;
}

unsigned char nodeID() {
  return NODE_NAME[0];
}
```

# File: mac.h

```
#ifndef MAC_H
#define MAC_H
// -*- Mode: C++ -*-
//
// File:        mac.h
// Description: header file for Medium Access support
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.


#include "pkt.h"

// void mac_task(void) _task_ MAC_TASK;

void mac_startTimer();
// Grab TMR1 and use it to count down gBackoffCounter for MAC timing.
// TMR1 is "stolen" by the radio whenever a packet is actively being
// transmitted or received, so counting stops when the radio is active.
// (This is a feature, not a bug.)


void mac_xmitPkt(pkt_t xdata *pkt);
// queue a packet for subsequent transmission by the MAC task.  Called
// by anybody that wishes to send a packet, but normally called from
// the GRAd thread.

void mac_recvPkt(pkt_t xdata *pkt);
// Pass a packet to the MAC layer.  If GRAD runs in a separate thread,
// this will queue the packet until GRAd fetches it.  Otherwise, it is
// passed directly to grad_recvPkt() for processing.

#endif
```

# File: mac.c

```
// -*- Mode: C++ -*-
//
// File:        mac.c
// Description: medium access.  defer transmission until presumed safe.
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// ============================================================
// queue for outgoing and incoming packets

#include "mac.h"

#include "arbor.h"
#include "stats.h"
#include "vector.h"     // for DEFINE_VECTOR()...
#include <aduc824.h>    // for register defs
#include <rtx51tny.h>   // for K_SIG, etc
#include <stdlib.h>     // for rand()
#include "rad.h"        // rad_xmitPkt()...
#include "appr.h"       // for appr_didXmit()
#include "screen.h"
```

```
#define MAC_BACKOFF_TICS (0x10000 - 5000)
// non-essential timing: we want to generate a TRM1 ISR once every
// 5 mSec, or once every 5000 TMR1 tics, not counting when the
// radio is in use

// queue for packets waiting to be transmitted by MAC_TASK
#define MAC_QUEUE_SIZE 10
DEFINE_VECTOR(gMacQueue, MAC_QUEUE_SIZE);

// ================================================================
// ================================================================
// binary exponential backoff

// counts down whenever radio is idle.  When it hits zero, we send
// the next packet.
unsigned char gBackoffCounter;

static unsigned char gBackoffExponent;
#define MAX_BACKOFF 8

static unsigned char const _masks[MAX_BACKOFF] = {
  0x07, 0x0f, 0x1f, 0x3f, 0x7f, 0x7f, 0x7f, 0x80
};

static unsigned char const _mindly[MAX_BACKOFF] = {
  0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x80
};

// gBackoffExponent is the exponent of the backoff.  The range of possible
// backoff values returned by backof() looks something like this:
// dly: 012345678901234567890123456789012345678901234567890123456789012345678
// g=0:   --------
// g=1:     ----------------
// g=2:         --------------------------------
// g=3: ...
static unsigned char generateBackoff() {
  // compute a random delay, measured in tics, according to the
  // current backoff exponent.
  unsigned char dly;
  dly = _mindly[gBackoffExponent] + (rand() & _masks[gBackoffExponent]);
  stats_backoff(dly);           // log longest backoff
  return dly;
}

static void resetBackoff() {
  gBackoffExponent = 0;
}

static void incrementBackoff() {
  if (gBackoffExponent != MAX_BACKOFF-1) gBackoffExponent++;
}

#if 0
static void decrementBackoff() {
  if (gBackoffExponent != 0) gBackoffExponent--;
}
#endif
```

```
// ================================================================
// ================================================================
// MAC routines

// ================================================================
// ================================================================
// MAC transmit thread.
// loop: [1] wait for a packet to appear in gMacQueue
//       [2] set backoff counter, wait for it to count down
//       [3] if packet is still in gMacQueue, transmit it
//       [4] loop
// The actual transmission is handled in this thread.
//
void mac_task(void) _task_ MAC_TASK {

  // one-time initialization
  vector_init(VECTOR(gMacQueue), MAC_QUEUE_SIZE);
  resetBackoff();
  gBackoffCounter = 0;
  mac_startTimer();

  while (1) {
    pkt_t xdata *pkt;
    // Wait for a packet to become available.  Don't remove from the
    // queue it until after we've waited for the backoff interval,
    // since some other thread might wish to prune the packet from the
    // queue in the interim.
    SCREEN_TASK(("mac_task(1)"));

    while (vector_count(VECTOR(gMacQueue)) == 0) {
      resetBackoff();
      os_wait2(K_SIG, 0);       // mac_xmitPkt() will generate signal
    }

    // Initialize the backoff counter according to the current backoff
    // exponent and then wait until it counts down to zero
    gBackoffCounter = generateBackoff();
    while (gBackoffCounter > 0) {
      // tmr1_interrupt() will signal us when gBackoffCounter hits 0
      os_wait2(K_SIG, 0);
    }

    // backoff has expired.  If there's still a packet available,
    // format and transmit the packet.  Tell APP that the packet
    // was sent, then free it.
    if ((pkt = vector_dequeue(VECTOR(gMacQueue))) != NULL) {
      rad_xmitPkt(pkt);
      SCREEN_TASK(("mac xmt: ")); PKT_PRINT(pkt);
      appr_didXmit(pkt);
      pkt_free(pkt);
    }
  }
}

void tmr1_interrupt(void) interrupt 3 using 2 {
  // Called regularly whenever TMR1 is configured as the MAC backoff
  // counter, namely, whenever the radio isn't sending or receiving.
  // This has the effect that gBackoffCounter only decrements
```

```
  // when the airwaves are idle.  When gBackoffCounter hits zero,
  // this ISR sends a signal to the MAC task.
  TH1 = MAC_BACKOFF_TICS>>8;    // reload TMR1
  TL1 = MAC_BACKOFF_TICS&0xff;  // ...
  if (gBackoffCounter > 0) {
    if (--gBackoffCounter == 0) isr_send_signal(MAC_TASK);
  }
}

void mac_startTimer() {
  // Configure TMR1 to count down MAC backoff tics, generating an
  // interrupt once every 1 mSec.
  //
  // see also rad_startTimer() in rad.c
  TR1 = 0;          // stop running
  TMOD &= 0x0f;     // clear bits for TMR1
  TMOD |= BITMASK(0,0,0,1,0,0,0,0); // 16 bit mode for TMR1
  TH1 = MAC_BACKOFF_TICS>>8;    // setup reload value
  TL1 = MAC_BACKOFF_TICS&0xff;  // ...
  TR1 = 1;          // start running
  ET1 = 1;          // enable TMR1 interrupts
}


// ================================================================
// queue a packet for subsequent transmission by the MAC task
// Called by anybody that wishes to send a packet, but normally
// called from the application thread.  If the queue fills up,
// throw away the oldest.

void mac_xmitPkt(pkt_t xdata *pkt) {

  // ## don't call until MAC_TASK has been started

  pkt_free(vector_shove(VECTOR(gMacQueue), pkt));
  // stats_macQueuePkt(vector_count(VECTOR(gMacQueue)));
  if (rad_isBusy()) incrementBackoff();
  os_send_signal(MAC_TASK);     // tell mac that a packet is waiting
}
```

# File: pkt.h

```
// -*- Mode: C++  -*-
#ifndef PKT_H
#define PKT_H
// -*- Mode: C++ -*-
//
// File:        pkt.h
// Description: support for packets, segments and payloads
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.


// A packet carries information among nodes.  A packet has an external
// and an internal representation.  The external form of a packet is a
// serial stream of bytes, passed to the radio.  The internal form is
// a linked list of pkt structures.  Each pkt structure corresponds to
// an abstraction layer, and can be conveniently appxed onto the head
// of the pkt list when generated or popped from the list when
// received.

// payload size determined by size of largest packet, currently the
// cost table reports.
#define MAX_PAYLOAD_SIZE 16

// Each packet segment carries a type with it.  Perhaps this is not
// the cleanest abstraction, but all the specific packet types are
// listed here.
typedef enum {
  SEG_TYPE_EOP = 0,
```

```
  SEG_TYPE_GRADIENT,    // 01 payload contains gradient routing info
  SEG_TYPE_DISCOVERY,   // 02 payload contains discovery routing request
  SEG_TYPE_TEXT,        // 03 payload contains text
  SEG_TYPE_COST_L,      // 04 payload contains low half of cost table
  SEG_TYPE_COST_H,      // 05 payload contains high half of cost table
  SEG_TYPE_STATS,       // 06 payload contains logging statistics
  SEG_TYPE_ADC,         // 07 payload contains analog readings
  SEG_TYPE_ARQ,         // 08 payload contains request for reply
  SEG_TYPE_ACK,         // 09 payload contains reply
  SEG_TYPE_APPX,        // 0a payload contains params for appx process
  SEG_TYPE_PING,        // 0b ping packet
  SEG_TYPE_TIME,        // 0c time report
  MAX_SEG_TYPE
} seg_type;

typedef struct _pkt_t {
  struct _pkt_t xdata *fNext;
  seg_type fType;
  unsigned char fSize;
  unsigned char fPayload[MAX_PAYLOAD_SIZE];
} pkt_t;

// initialize the packet system
void pkt_init();

pkt_t xdata *pkt_alloc();
// allocate a single packet segment

void pkt_free(pkt_t xdata *head);
// free a chain of packet segments

pkt_t xdata *pkt_copy(pkt_t xdata *pkt);
// make a "deep copy" of pkt

#define pkt_type(p) ((p)->fType)
// get/set the packet type for this segment
// seg_type pkt_getType(pkt_t xdata *pkt);
// void pkt_setType(pkt_t xdata *pkt, seg_type type);

#define pkt_size(p) ((p)->fSize)
// get/set the number of bytes in the payload.
// unsigned char pkt_getSize(pkt_t xdata *pkt);
// void pkt_setSize(pkt_t xdata *pkt, unsigned char size);

#define pkt_next(p) ((p)->fNext)
// get/set the next packet in the list of packets
// pkt_t xdata *pkt_getNext(pkt_t xdata *pkt);
// void pkt_setNext(pkt_t xdata *pkt, pkt_t xdata *next);

#define pkt_payload(p) ((p)->fPayload)
// reference the first byte of the payload
// unsigned char xdata *pkt_getPayload(pkt_t xdata *pkt);

pkt_t xdata *pkt_find_segment(pkt_t xdata *pkt, seg_type type);
// find a packet segment of the given type in the packet
// chain, returning NULL if not found

void pkt_print(pkt_t xdata *seg);
// print one segment in hex

void pkt_dumpHex(pkt_t xdata *pkt);
// print a string of segments in hex

#endif
```

# File: pkt.c

```
// -*- Mode: C++ -*-
//
// File:        pkt.c
// Description: support for packets, segments and payloads
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "pkt.h"
#include "arbor.h"
#include "constell.h"
#include <stdio.h>
#include <string.h>
#include "screen.h"
#include "grad.h"
#include "arq.h"
#include "appx.h"

// ============================================================
// managing packets
```

```
// a global pool of pkt structures
static pkt_t xdata * data gFree = NULL;
static unsigned char gAvail = 0;        // debug

#define MAX_PACKETS 30
static pkt_t xdata _pkts[MAX_PACKETS];

static char _is_valid(pkt_t xdata *p) {
  return (_pkts <= p) && (p < &_pkts[MAX_PACKETS]);
}

static void _pkt_free_seg(pkt_t xdata *seg) {
  // return a single packet segment to the freelist
  // PRINTF(("pkt_fs(%x) ", (short)seg));
  if (!_is_valid(seg)) return;
  gAvail++;
  seg->fNext = gFree;
  gFree = seg;
  SCREEN(("Avail=%2bu", gAvail));
}

void pkt_init() {
  // set up the pool of packet structures
  int i=MAX_PACKETS;
  gFree = NULL;
  gAvail = 0;
  while (i--) {
    _pkt_free_seg(&_pkts[i]);
  }
}

pkt_t xdata *pkt_alloc() {
  // pop a packet segment from the freelist
  pkt_t xdata *p;

  // PRINTF(("pkt_a() => %x ", (short)gFree));
  if (gFree == NULL) {
    printf("no more packets");
    while (1);
  }
  p = gFree;
  gFree = p->fNext;
  if (!_is_valid(gFree)) {
    printf("freelist clobbered");
    while (1) ;
  }
  gAvail--;
  SCREEN(("avail=%2bu", gAvail));
  p->fNext = (short)0;
  return p;
}

// free the entire packet chain headed by head.
void pkt_free(pkt_t xdata *seg) {
  while (seg) {
    pkt_t xdata *next = pkt_next(seg);
    _pkt_free_seg(seg);
    seg = next;
  }
}

// make a deep copy of a packet chain
pkt_t xdata *pkt_copy(pkt_t xdata *pkt) {
  pkt_t xdata *first = NULL;
  pkt_t xdata *prev = NULL;

  while (pkt != NULL) {
    pkt_t xdata *seg = pkt_alloc();
    // I could be clever and only copy the part of the payload that's
    // active, but ...
    memcpy(seg, pkt, sizeof(pkt_t));
    if (prev == NULL) {
      first = seg;
    } else {
      pkt_next(prev) = seg;
    }
    prev = seg;
    pkt = pkt_next(pkt);
  }
  return first;
}

pkt_t xdata *pkt_find_segment(pkt_t xdata *pkt, seg_type type) {
  // find a packet segment of the given type in the packet
  // chain, returning NULL if not found
  while (pkt != NULL) {
    if (pkt_type(pkt) == type) return pkt;
    pkt = pkt_next(pkt);
  }
  return NULL;
}

// since we're sending raw binary over the serial port, the
// author of the server wanted a little error check to help
// stay in sync.  Each segment count byte has the high order
// bit turned on, each packet size byte has the high two order
// bits turned on.
#define SEG_COUNT_FLAG 0x80
#define PKT_SIZE_FLAG 0xC0

static unsigned char pkt_countsegs(pkt_t xdata *pkt) {
  unsigned char i = 0;
  while (pkt != NULL) {
    i++;
    pkt = pkt_next(pkt);
  }
  return i;
}

static unsigned char code toHex[16]={
  '0','1','2','3','4','5','6','7',
  '8','9','a','b','c','d','e','f'};


static void puthex(unsigned char ch) {
  putchar(toHex[ch>>4]);
```

```
  putchar(toHex[ch&0x0f]);
}

void pkt_print(pkt_t xdata *seg) {
  unsigned char i = seg->fSize;
  char *p = pkt_payload(seg);
  puthex(i+1);
  puthex(seg->fType);
  while (i--) {
    puthex(*p++);
  }
}

void pkt_dumpHex(pkt_t xdata *pkt) {
  // dump pkt to serial port in sligtly formatted hex form
  printf("\n");
  puthex(pkt_countsegs(pkt));
  while (pkt != NULL) {
    printf("\n ");
    pkt_print(pkt);
    pkt = pkt_next(pkt);
  }
}
```

# File: rad.h

```
#ifndef RAD_H
#define RAD_H
// -*- Mode: C++ -*-
//
// File:        rad.h
// Description: low-level radio support
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "pkt.h"

void rad_init();
// cold start for the radio

#if 0
void rad_printBuffer();
#endif

bit rad_isBusy();
// returns true whenever the radio is actively transmitting or
// receiving.

void rad_xmitPkt(pkt_t xdata *pkt);
// Transmit contents of packet immediately.  Kills RADR_TASK thread,
// handles transmission in caller's thread, the restarts RADR_TASK.
// NOT to be called from the RADR_TASK thread.

pkt_t xdata *rad_recvPkt();
```

```
// Block until a buffer of data has been received.  Parse the buffer
// into a pkt_t structure and (if CRC is valid) return it.  If CRC is
// invalid, returns NULL.  MUST be called from within RADR_TASK

void rad_standby();
// shut down radio

#endif
```

# File: rad.c

```
// -*- Mode: C++ -*-
//
// File:        rad.c
// Description: low-level I/O support for BART and TR1000 radio
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "rad.h"
#include "pkt.h"

#include "arbor.h"
#include <rtx51tny.h>   // for os_wait()...
#include <aduc824.h>    // for register defs
#include "constell.h"   // for LEDs
#include "id.h"                 // for nodeID()
#include "stats.h"
#include "sync.h"
#include "appr.h"       //  for appr_recvPkt() ...
#include "mac.h"        // for mac_startTimer()
#include "screen.h"
#include <stdio.h>

#define BART_HOLDOFF_TICS (256-35)
// essential timing: BART needs BART_CLK stable for 35 uSec after each
// transition.  Assuming a 12MHz system clock and a 1 uSec cycle time,
// TMR1 must roll over once every 35 tics.

#define AWAIT_BART()    os_wait2(K_TMO | K_SIG, 1)
// #define AWAIT_BART() os_wait2(K_SIG, 0)

#define BART_IS_READY() (!BART_READY)

// Set the holdoff timer counting.  TF1 will be set at the end of
// the holdoff period.
#define RESTART_HOLDOFF()       \
  TL1 = BART_HOLDOFF_TICS;      \
  TF1 = 0

// Busy wait until holdoff flag is set.  Assumes RESTART_HOLDOFF()
// has been invoked previously
#define AWAIT_HOLDOFF()         while (!TF1)

// ================================================================
// Local function prototypes

static void _rad_setRecvMode();
// configure the radio (and bart chip, and timers, and data lines)
// for receive mode.

static void _rad_recvBuffer();
// configure radio to receive, await a packet of data, and read it
// into this module's internal buffer

static unsigned char _rad_getByte();
// get a single byte from the radio (via the BART chip)

static pkt_t xdata *rad_import();
// convert the "raw" contents of the radio's internal buffer
// into a linked packet structure.  Return NULL for bad packet.

void rad_export(pkt_t xdata *pkt);
// copy and convert the contents of a linked packet structure
// into "raw" format in the radio's internal buffer.

static void _rad_xmitBuffer();
// configure the radio to transmit, send a packet of data from this
// module's internal buffer, reconfigure radio to receive

static void _rad_putByte(unsigned char b);
// write a single byte to the radio (via the BART chip)

static void _rad_setBARTXmit();
// configure the bart chip for transmit mode.

static void _rad_finishBARTXmit();
// finish any transmit in progress, then configure bart chip for receive.
```

```
typedef short crc_t;
static crc_t gCRC;
#define _crc_state() gCRC
static void _crc_init();
static unsigned char _crc(unsigned char c);
unsigned char hexToNibble(unsigned char ch);


// ===============================================================
// Radio "staging" buffer

// The radio requires high-speed data transfer into and out of a
// dedicated buffer.  The STAGING_AREA macro defines in which
// segment this buffer resides

// #define STAGING_IN_PDATA
// STAGING_IN_PDATA doesn't work.  Every packet sent and received
// appears to have a bad checksum.  Without resorting to a 'scope,
// my guess is that data transfers between pdata and the radio are
// slow, resulting in under- and over-runs.

// STAGING_BUFFER_SIZE is hand chosen to fill out the area without
// overflowing it.  It determines the maximum number of bytes that
// may be transmitted in a radio packet.  This controls the number
// of byte in data space dedicated to a "staging" buffer for the
// low-level radio I/O.  There are only 128 bytes of data space available
// in the system, so if choosing STAGING_BUFFER_SIZE is a tradeoff: too
// small and packet size is limited.  Too large, data space is eaten up.

#ifdef STAGING_IN_PDATA
#define STAGING_AREA pdata
#define STAGING_BUFFER_SIZE 0xfe
#else
#define STAGING_AREA data
// #define STAGING_BUFFER_SIZE 43
#define STAGING_BUFFER_SIZE 43
#endif

// Dedicated buffer in data space for radio I/O.  See note in
// _rad_recvBuffer() regarding the +1 .
static unsigned char STAGING_AREA sRadBuf[STAGING_BUFFER_SIZE + 1];

// place to cache RTC clock values at the onset of receiving a
// packet.  Used as part of the time synchronization process.
static vtime_t gRecvTime;

// ===============================================================
// TMR1 management

// When start_macTimer() is called, it sets up TMR1 to
// interrupt once every millisecond.  The TMR1 interrupt
// service decrements gMACBackoffCounter, and if it hits
// zero, signals the MAC task.
//
// When start_radio_timer() is called, it sets up TMR1 to
// set its flag every 35 uSec, and disables its interrupt
// flag to inhibit the calling of the interrupt service.
//
// This has the properties we want: when the radio is
// busy sending or receiving data, we want to inhibit
// the counting down of the gMACBackoffCounter.  This
// is an approximate implementation of the 802.11 style
// MAC layer.

bit gRadIsBusy;

static void rad_startTimer() {
  // Configure TMR1 to auto-reload once every BART_HOLDOFF_TICS,
  // disable interrupts.
  ET1 = 0;          // disable TMR1 interrupts
  TR1 = 0;          // stop running
  TMOD &= 0x0f;     // clear bits for TMR1
  TMOD |= BITMASK(0,0,1,0,0,0,0,0); // auto reload (mode 2) for TMR1
  TH1 = BART_HOLDOFF_TICS;  // setup reload value
  TR1 = 1;              // start running
  RESTART_HOLDOFF();
}

// ===============================================================
// initialization and interrupt code

void rad_init() {
  TR1000_CTL_DIRECTION = BITMASK(0,0,0,0,0,1,1,0); // setup mode ctl pins
  TR1000_CTL_DRIVE = 0x00;      // standard CMOS I/O
  BART_DATA_CONTROL = 0x00;     // for pins PB0 - PB7
  BART_DATA_DRIVE = 0x00;       // standard CMOS I/O

  // enable external interrupts on INT0, edge trigger
  IT0 = 1;                      // INT0 edge triggered
  EX0 = 1;                      // enable INT0 interrupts
  _rad_setRecvMode();           // configure for receive mode
}

void int0_interrupt(void) interrupt 0 using 2 {
  // Notify the RADR_TASK that BART_READY has come true.  Note that
  // INT0 interrupts arrive as interrupt 0.  This routine uses
  // register bank 2 to avoid copying registers.
  isr_send_signal(RADR_TASK);
}

bit rad_isBusy() {
  return gRadIsBusy;
}

#if 0
void rad_printBuffer() {
  unsigned char i;
  for (i = 0; i<RAD_BUF_SIZE; i++) {
    printf("%02bx ", sRadBuf[i]);
    if ((i+1)%16 == 0) puts("");
  }
}
#endif

// ===============================================================
```

```
// ===================================================================
// Receiving Data
//

// ===================================================================
// Receive Task
//
// Repeatedly try to read a packet from the radio.  When a valid
// packet is found, call mac_recvPkt() to process it.

void radr_task() _task_ RADR_TASK {
  pkt_t xdata *p;

  while(1) {
    SCREEN_TASK(("radr_task(1)"));
    p = rad_recvPkt();
    if (p != NULL) {
      PKT_PRINT(p);
      stats_goodRecvPkt();      // note a good packet
      appr_recvPkt(p);
    } else {
      stats_badRecvPkt();       // not a good packet.
    }
  }
}

pkt_t xdata *rad_recvPkt() {
  pkt_t xdata *pkt;
  pkt_t xdata *pingSeg;

  // block (in _rad_recvBuffer()) until a low-level buffer of data
  // has been received.  Parse the buffer into a pkt_t structure
  // and (if CRC is valid) return it.  If CRC is invalid, returns
  // NULL.
  _rad_recvBuffer();
  pkt = rad_import();

  // If there is a PING segment in the received packet, fill in the
  // received time field with the onset time of reception, gRecvTime,
  // as captured by _rad_recvBuffer();
  if ((pingSeg = pkt_find_segment(pkt, SEG_TYPE_PING)) != NULL) {
    pingPayload xdata *pp = pkt_payload(pingSeg);
    pp->fTimeR = gRecvTime;
  }
  return pkt;
}

static void _rad_setRecvMode() {
  // configure radio, BART, and TMR1 for receiving radio data
  rad_startTimer();             // momentarily steal TMR1
  gRadIsBusy = 1;
  _rad_setBARTXmit();           // force BART into known mode
  _rad_finishBARTXmit();        //   ...
  TR1000_CTL0 = 1;              // set TR1000 to receive mode
  TR1000_CTL1 = 1;
  mac_startTimer();             // resume counting MAC backoff tics
  gRadIsBusy = 0;
}

static void _rad_recvBuffer() {
  // fetch a packet of data from the radio into sRadBuf[].
  // [1] Force the BART chip to search for new sync header.  BART
  //     (in _rad_getByte()) will return the first character found
  //     after a sync header.
  // [2] Read the byte.  If it is not RAD_PKT_LEADER, goto [1].
  // [3] Read in a series of segments.  Each segment starts with
  //     a byte count, followed by that many bytes of data.  The
  //     subsequent byte is the byte count for the next segment.
  //     A byte count of zero terminates the chain.
  // [4] Return
  unsigned char data len;
  unsigned char STAGING_AREA * data p;
  unsigned char STAGING_AREA * data pEnd;

  LED_OFF(RED_LED);         // clear bad pkt indicator

  p = sRadBuf;
  pEnd = &sRadBuf[STAGING_BUFFER_SIZE];

  _rad_setRecvMode();

  while (!BART_IS_READY()) {
    AWAIT_BART();
  }
  rad_startTimer();             // got first char, stop counting mac tics
  gRadIsBusy = 1;

  gRecvTime = sync_getLocalTime(); // capture time at onset of reception

  LED_ON(ORANGE_LED);       // actively receiving
  len = _rad_getByte();

  while (len != 0) {
    // this loop can overshoot the buffer length by one, so we've
    // made the buffer one extra byte long...
    *p++ = len;
    while ((len--) && (p<pEnd)) {
      *p++ = _rad_getByte();
    }
    if (p >= pEnd) break;
    len = _rad_getByte();
  }
  *p = 0;                       // write terminating byte

  LED_OFF(ORANGE_LED);          // no longer receiving
  AWAIT_HOLDOFF();
  mac_startTimer();             // start counting MAC backoff again
  gRadIsBusy = 0;
}

static unsigned char _rad_getByte() {
  // receive a byte from the radio via the BART interface.  If the
  // BART receive buffer is empty, the caller's thread will be
  // blocked.  Assumes the radio is in receive mode.
  unsigned char b;
  AWAIT_HOLDOFF();              // buzz until prior holdoff elapses
```

```
  // BART_READY line is now guaranteed to valid.  Check its value,
  // block this thread if BART's input FIFO is empty.
  while (!BART_IS_READY()) {
    AWAIT_BART();
  }

  BART_CLOCK = 1;               // request the data
  RESTART_HOLDOFF();            // hold BART_CLOCK high for holdoff period
  AWAIT_HOLDOFF();              // ...

  b = BART_DATA_IN;             // data now valid.  latch it
  BART_CLOCK = 0;               // finish data transfer
  RESTART_HOLDOFF();            // hold BART_CLOCK low for holdoff period
  // next call to _rad_getByte() will complete the holdoff
  // AWAIT_HOLDOFF();

  return b;
}


static unsigned char _pkt_isReasonable(unsigned char len, unsigned char
type) {
  // try to filter out stupid packets before we allocate storage for them
  //
  if ((len == 0) || (len > MAX_PAYLOAD_SIZE+1)) return 0;
  if ((type == SEG_TYPE_EOP) || (type == 0x88)) return 0;
  return 1;
}

static pkt_t xdata *rad_import() {
  // read the radio's raw data buffer into pkt structures,
  // perform checksumming, etc.  Return a packet structure
  // if the data is intact, NULL otherwise
  unsigned char STAGING_AREA *src;
  unsigned char len, type;
  unsigned short crcFound;
  unsigned char xdata * data pay;
  pkt_t xdata * data pkt;
  pkt_t xdata * data prev;
  pkt_t xdata * data head;

  src = &sRadBuf[0];
  _crc_init();
  head = prev = NULL;

  while (1) {
    len = _crc(*src++);
    type = _crc(*src++);
    if (!_pkt_isReasonable(len,type)) break;
    len--;      // account for type field just read.
    pkt = pkt_alloc();
    pkt_size(pkt) = len;
    pkt_type(pkt) = type;
    pay = pkt_payload(pkt);
    while (len--) {
      *pay++ = _crc(*src++);
    }
    // link this packet into chain of packets
    if (prev == NULL) {
      head = pkt;
    } else {
      pkt_next(prev) = pkt;
    }
    prev = pkt;
  }

  crcFound = 0;
  if ((type == SEG_TYPE_EOP) && (len == 5)) {
    // good prospects.  Read in last 4 bytes as hex chars,
    // compare against accumulated CRC value
    for (len = 0; len < 4; len++) {
      crcFound <<= 4;
      crcFound += hexToNibble(*src++);
    }
  }

  if (_crc_state() == crcFound) {
    LED_OFF(RED_LED);
    return head;                // a valid packet!
  } else {
    LED_ON(RED_LED);    // bad pkt indicator
    pkt_free(head);             // release bogus segments
    return NULL;
  }
}

// ================================================================
// ================================================================
// Transmitting Data
//

void rad_xmitPkt(pkt_t xdata *pkt) {
  // Transmit a packet immediately.  NOT to be called from within the
  // RADR_TASK thread.
  pkt_t xdata *pingSeg = pkt_find_segment(pkt, SEG_TYPE_PING);

  SCREEN_TASK(("rad_xmitPkt(1)"));

  os_delete_task(RADR_TASK);    // stop the receiver

  // If the packet contains a SEG_TYPE_PING segment, copy the
  // nodeID and local time into it just prior to transmission.
  if (pingSeg != NULL) {
    pingPayload xdata *pp = pkt_payload(pingSeg);
    pp->fNodeID = nodeID();
    pp->fTimeX = sync_getLocalTime();
  }

  SCREEN_TASK(("rad_xmitPkt(2)"));
  rad_export(pkt);              // copy in to data-space buffer
  _rad_xmitBuffer();            // blurt
  // MAC is responsible for freeing the packet
  // pkt_free(pkt);             // free the packet
  SCREEN_TASK(("rad_xmitPkt(3)"));
  os_create_task(RADR_TASK);    // restart receiver
```

```
}


void rad_export(pkt_t xdata *pkt) {
  // write the contents of the pkt structures headed by pkt into the
  // radio's low-level buffer, complete with checksum.
  unsigned char STAGING_AREA *dst;
  unsigned char len;
  unsigned char xdata *pay;

  dst = &sRadBuf[0];
  _crc_init();
  while (pkt) {
    len = pkt_size(pkt);
    *dst++ = _crc(len + 1);     // payload + type
    *dst++ = _crc(pkt_type(pkt));
    pay = pkt_payload(pkt);
    while (len--) {
      *dst++ = _crc(*pay++);
    }
    pkt = pkt_next(pkt);
  }
  // output a final segment with len = 5, type = EOP, payload = CRC
  // as a hex string, and a final terminating null
  *dst++ = _crc(5);
  *dst++ = _crc(SEG_TYPE_EOP);
  sprintf(dst, "%04x\0", _crc_state());

  // Now rad_buffer() has been loaded up.  Send it...
  // rad_xmitBuffer();
  dst = &sRadBuf[0];
}


static void _rad_xmitBuffer() {
  // send the contents of sRadBuf[] to the radio.  sRadBuf[] is
  // expected to consist of one or more packets.  Each packet starts
  // with a byte count, followed by that many bytes of data.  The next
  // byte following is the number of bytes in the subsequent packet.
  // A byte count of zero terminates the chain.  The terminating zero
  // is transmitted.
  unsigned char len;
  unsigned char STAGING_AREA *p = sRadBuf;

  rad_startTimer();             // grab TMR1 for radio (no mac tics)
  gRadIsBusy = 1;
  _rad_setBARTXmit();
  TR1000_CTL0 = 0;              // configure TR1000 for ASK transmit
  TR1000_CTL1 = 1;              //    ...

  LED_OFF(RED_LED);         // clear bad pkt indicator
  LED_ON(AMBER_LED);        // indicate transmit mode
  while ((len = *p++) != 0) {
    _rad_putByte(len);
    while (len--) _rad_putByte(*p++);
  }
  _rad_putByte(0);              // transmit terminating byte

  // get the radio out of transmit mode gracefully...
  _rad_finishBARTXmit();        // wait for BART xmit fifo to drain
  TR1000_CTL0 = 1;              // switch TR1000 to receive mode
  TR1000_CTL1 = 1;

  LED_OFF(AMBER_LED);           // indicate end of transmit mode
  mac_startTimer();             // resume counting MAC backoff tics
  gRadIsBusy = 0;
}

static void _rad_putByte(unsigned char b) {
  // Send a byte to the radio via the BART interface chip.  If the
  // BART transmit buffer is full, the caller's thread will be
  // blocked.  Assumes the radio is in transmit mode.
  BART_DATA_OUT = b;            // set up the data on the i/o lines
  AWAIT_HOLDOFF();              // buzz until prior holdoff elapses

  // BART_READY line is now guaranteed to valid.  Check its value,
  // block this thread if BART's input FIFO is full.
  while (!BART_IS_READY()) {
    AWAIT_BART();
  }

  BART_CLOCK = 1;               // announce the data
  RESTART_HOLDOFF();            // hold BART_CLOCK high for holdoff period
  AWAIT_HOLDOFF();              // ...

  BART_CLOCK = 0;               // finish data transfer
  RESTART_HOLDOFF();            // hold BART_CLOCK low for holdoff period
  // next call to _rad_putByte() will complete the holdoff
  // AWAIT_HOLDOFF();
}

// ============================================================
// switching BART modes
//
// In BART_RECV_MODE, bart reads serial data from the radio and
// writes it to the parallel port.  BART_READY stays false until
// at least one byte is available in the fifo.
//
// In BART_XMIT_MODE, bart reads bytes from the parallel port and
// sends serial data to the radio.  BART_READY stays true unless
// the host fills up the fifo.

static void _rad_setBARTXmit() {
  // Configure BART to transmit data.  (More importantly, set
  // the BART chip into a well defined state.)  Upon exit:
  // - bart parallel port set to receive
  // - bart in transmit mode and ready to receive host data
  // If this routine is called while BART is already in transmit
  // mode, nothing particularly bad happens.  Note that this routine
  // doesn't set the radio control lines.
  BART_CLOCK = 0;
  BART_MODE = BART_MODE_XMIT;   // Tell BART to switch to xmit mode
  AWAIT_HOLDOFF();
  // ### SOME RADIOS SEEM TO GET HUNG HERE AT STARTUP.  WHY?
  while (!BART_IS_READY()) {    // wait until BART is ready
    AWAIT_BART();
```

```
  }
  BART_DATA_DIRECTION = XMIT;   // set data direction towards BART
}

static void _rad_finishBARTXmit() {
  // Tell BART to leave transmit mode and wait for any buffered data
  // it has in its fifo to transmit before returning.  Assumes that
  // BART has been in transmit mode, that TMR1 is set up as a holdoff
  // timer.

  if (BART_MODE == BART_MODE_RECV) {
    // quit now if BART is already configured in receive mode,
    // else the "while (!BART_IS_READY()) ..." below would hang.
    return;
  }

  AWAIT_HOLDOFF();
  BART_CLOCK = 0;

  // BART asserts the BART_READY line while the fifo is draining,
  // so we must handle the rare case that the fifo is full upon
  // entering _rad_finishBARTXmit() by letting the fifo get to a
  // non-full state before continuing...
  while (!BART_IS_READY()) {
    AWAIT_BART();
  }

  BART_DATA_DIRECTION = RECV;   // set data direction from BART
  BART_MODE = BART_MODE_RECV; // tell bart to enter receive mode
  // Unconventional: BART will assert BART_IS_READY() until its fifo
  // drains.  This is because in the recv mode, the BART_IS_READY()
  // is asserted when the fifo is EMPTY.
  while (BART_IS_READY()) {             // wait for BART's fifo to drain
    os_wait2(K_TMO, 0);                 // K_SIG won't work here...
  }

  // BART's transmit fifo is now empty.
}

// ================================================================
// CRC generation and checking
// The CRC polynomial is feeble but simple to compute...

static void _crc_init() {
  gCRC = 0xf1f1;
}

static unsigned char _crc(unsigned char c) {
  if (gCRC < 0) {
    gCRC = (gCRC << 1) + c + 1;
  } else {
    gCRC = (gCRC << 1) + c;
  }
  return c;
}

// ================================================================
// auxiliary routines

unsigned char hexToNibble(unsigned char ch) {
  if (ch <= '9') {
    return ch - '0';
  } else if (ch <= 'F') {
    return ch - 'A' + 10;
  } else if (ch <= 'f') {
    return ch - 'a' + 10;
  }
}
```

## File: screen.h

```
#ifndef SCREEN_H
#define SCREEN_H
// -*- Mode: C++ -*-
//
// File:        screen.h
// Description: diagnostic printout to VT100 compatible screen
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// de-comment the following line to enable the SCREEN_xxx macros.
// Displayed on a HyperTerm or equivalent communications program,
// this will display diagnostics: each thread state is printed on
// its own line.
// #define SCREEN_ENABLE

#ifdef SCREEN_ENABLE
#include <stdio.h>

void screen_cleareol(void);
void screen_clear(void);
void screen_goto(unsigned char row, unsigned char col);
void screen_task_prefix(void);

#define SCREEN_CLEAR() screen_clear()
#define SCREEN_TASK(string) screen_task_prefix();  printf string;
screen_cleareol();

#define SCREEN(string) \
  screen_goto(1, 1);    \
  printf string

#define PKT_PRINT(pkt) pkt_print(pkt)

#else // SCREEN_ENABLE

#define SCREEN_CLEAR()
#define SCREEN_TASK(string)
#define SCREEN(string)
#define PKT_PRINT(pkt)

#endif  // SCREEN_ENABLE


#endif  // ifdef SCREEN_H
```

## File: screen.c

```
// -*- Mode: C++ -*-
//
// File:        screen.c
// Description: diagnostic printout to VT100 compatible screens
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "screen.h"

#ifdef SCREEN_ENABLE
```

```
#include <rtx51tny.h>
#include <stdio.h>

static unsigned char gCount;

void screen_cleareol() {
  printf("%bc[0K", 0x1b);
}

void screen_goto(unsigned char r, unsigned char c) {
  printf("%bc[%bd;%bdH", 0x1b, r, c);
}

void screen_clear() {
  printf("%bc[2J", 0x1b);     // erase screen
  gCount = 0;
}

void screen_task_prefix() {
  unsigned char id = os_running_task_id();
  screen_goto(id+4, 1);
  printf("%2bd:%3bu: ", id, gCount++);
}

#endif // ifdef SCREEN_ENABLE
```

# File: serial.h

```
#ifndef SERIAL_H
#define SERIAL_H
// -*- Mode: C++ -*-
//
// File:        serial.h
// Description: header file for serial I/O routines
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

void serial_init(void);

bit serial_charIsAvailable();
// returns true if a char has been typed

// bit serial_hasInput();
// returns true if a line of text has been typed

// char *serial_input();
// returns the serial input buffer

#endif  // ifndef SERIAL_H
```

# File: serial.c

```
// -*- Mode: C++ -*-
```

```
//
// File:        serialc
// Description: hardware support for serial I/O on ADuC824 processor
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "arbor.h"
#include <aduc824.h>    // for register definitions

  // The processor xtal clock is 12582912.  As best as I can measure
  // and guess, the input to TH2 is xtal/32 = 393216.  Using TH2 in
  // auto reload mode, some reasonable for 256-T and the resulting
  // baud rates are:
  //
  //   T actual target   %err
  //  10 39321.6 38400  0.024
  //  20 19660.8 19200  0.024
  //  41  9590.6  9600 -0.001
  //  82  4795.3  4800 -0.001
  // 164  2397.7  2400 -0.001

#define BAUD_38400  10
#define BAUD_19200  20
#define BAUD_9600   41
#define BAUD_4800   82
#define BAUD_2400  164

void serial_init() {

  PCON |= BITMASK(1,0,0,0,0,0,0,0); // "double" baud rate
  SCON = BITMASK(0,1,0,1,0,0,0,0);  // mode 1, rcv enable, 8 bit
  // Use TMR2 for baud rate generation
  T2CON = BITMASK(0,0,1,1,0,0,0,0); // Rx,Tx BRG, timer, auto-reload
  RCAP2H = 0xff;
  RCAP2L = 256-BAUD_38400;          // set baud rate

  TR2 = 1;                          // run the clock
  TI = 1;                           // ready to send first char
}

bit serial_charIsAvailable() {
  // return true if a character is in the input buffer
  return RI;
}
```

## File: stats.h

```
#ifndef STATS_H
#define STATS_H
// -*- Mode: C++ -*-
//
// File:        stats.h
// Description: gather statistics on packet-level I/O
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "pkt.h"

typedef struct _statsPayload {
  unsigned int fGoodRecv;       // # of good packets received
  unsigned int fBadRecv;        // # of bad packets received
  unsigned int fOrig;           // # of packets originated
  unsigned int fRelay;          // # of packets relayed
  unsigned int fFlood;          // # of discovery packets
  unsigned int fARQs;           // # of acks requested
  unsigned int fACKs;           // # of acks received
  unsigned char fMaxBackoff;    // max backoff generated
} statsPayload;

void stats_reset();
// reset the statistics counters

void stats_goodRecvPkt();
// note a valid packet received by the radio

void stats_badRecvPkt();
// note a packet with bad CRC received by the radio

void stats_origPkt();
// note the origination of a packet by GRAD

void stats_relayPkt();
// note the relaying of a packet by GRAD

void stats_floodPkt();
// note the origination of a discovery packet by GRAD

void stats_arq();
// note the sending of an ARQ

void stats_ack();
// note the reception of an ACK

void stats_backoff(unsigned char backoff);
// note the highest backoff seen

pkt_t xdata *stats_report(pkt_t xdata *next);
// create stat report packet

#endif
```

## File: stats.c

```
// -*- Mode: C++ -*-
//
// File:        stats.c
// Description: gather and report statistics on packet level I/O
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
```

```
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include <stdio.h>
#include <string.h>
#include "pkt.h"
#include "stats.h"

static statsPayload sStats;

#define MAX_INT ((1<<(sizeof(int)*8))-1)

static _inc_clamp(int v) {
  if (v == MAX_INT) {
    return v;
  } else {
    return v+1;
  }
}

#define INCREMENT(i) i = _inc_clamp(i)

void stats_reset() {
  // reset the statistics counters
  memset(&sStats, 0, sizeof(statsPayload));
}

void stats_goodRecvPkt() {
  // note a valid packet received at the radio level
  INCREMENT(sStats.fGoodRecv);
}

void stats_badRecvPkt() {
  // note a packet at the radio level with bad CRC
  INCREMENT(sStats.fBadRecv);
}

void stats_origPkt() {
  // note the origination of a packet by GRAD
  INCREMENT(sStats.fOrig);
}

void stats_relayPkt() {
  // note the relaying of a packet by GRAD
  INCREMENT(sStats.fRelay);
}

void stats_arq() {
  // note the sending of an ARQ
  INCREMENT(sStats.fARQs);
}

void stats_ack() {
  // note the reception of an ACK
  INCREMENT(sStats.fARQs);
}

void stats_floodPkt() {
  // note the origination of a discovery packet by GRAD
  INCREMENT(sStats.fFlood);
}

void stats_backoff(unsigned char backoff) {
  // note increased backoff, track highest seen
  if (backoff > sStats.fMaxBackoff) sStats.fMaxBackoff = backoff;
}

pkt_t xdata *stats_report(pkt_t xdata *next) {
  // create stat report packet
  pkt_t xdata *pkt = pkt_alloc();
  pkt_type(pkt) = SEG_TYPE_STATS;
  pkt_size(pkt) = sizeof(statsPayload);
  pkt_next(pkt) = next;
  memcpy(pkt_payload(pkt), &sStats, sizeof(statsPayload));
  return pkt;
}
```

## File: sync.h

```
// -*- Mode: C++ -*-
//
// File:        sync.h
// Description: support for decentralized synchronization
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#ifndef SYNC_H
#define SYNC_H

#include "pkt.h"

// support for the real time clock and synchronization among nodes.


// a virtual day for the system is defined as one minute,
// measured in 128ths of a second.  Time is taken modulo
// 7680.

#define VIRTUAL_DAY (60 * 128)
#define VIRTUAL_NOON (30 * 128)

typedef unsigned int vtime_t;

// A packet flagged SEG_TYPE_PING carries synchronization info.
// The fTimeX field is filled in (at the MAC level) just before
// the packet is transmitted.  The fTimeR field is filled in
// with the ping that the packet started arriving.

typedef struct _pingPayload {
  unsigned char fNodeID;        // sending node id
  vtime_t fTimeX;               // sender's time
  vtime_t fTimeR;               // receiver's time
} pingPayload;

// A packet flagged SEG_TYPE_TIME carries information on the
// real time clock and timing of the node relative to other
// nodes.
typedef struct _timePayload {
  vtime_t fLocalTime;           // this node's time
  vtime_t fMaxErr;              // maximum error seen recently
  int fSyncSent;                // # of sync packets sent
  int fSyncRcvd;                // # of sync packets received
} timePayload;

void sync_reset();
// reset the sync statistics counters

void sync_setPingInterval(unsigned char tenths);
// set the ping interval.

vtime_t sync_getLocalTime();
// return the current real time for this node.

void sync_serviceSeg(pkt_t xdata *pingSeg);
// called when a segment arrives.  Computes the error between the
// xmit ping and the recv ping and adjusts the clock accordingly.

// void sync_task(void) _task_ SYNC_TASK;
// Periodically transmits a ping packet to node within range.

pkt_t xdata *sync_report(pkt_t xdata *next);
// Return a SEG_TYPE_TIME segment containing timing info for this
// node.

#endif
```

## File: sync.c

```
// -*- Mode: C++ -*-
//
// File:        sync.c
// Description: support for decentralized synchronization
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
```

```
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "arbor.h"
#include "constell.h"
#include "mac.h"
#include "sync.h"
#include "pkt.h"
#include "screen.h"
#include <aduc824.h>    // for register definitions
#include <limits.h>     // for UCHAR_MAX
#include <rtx51tny.h>   // for os_wait()...
#include <stdlib.h>     // rand()
#include <string.h>     // memset()

/*
Ruminations on timing:

Time is measured by reading the internal real time clock and adding an
offset to it to form "local time."  Adjustments to the clock are made
by modifying the offset, not by resetting the real time clock.

Local time has a resolution of 1/128 seconds (determined by the real
time clock hardware), and rolls over once every 60 seconds, or 7680
tics (where 1 tic = 1/128 second).  Thus a "VIRTUAL_DAY" is defined as
7860 tics.

When a ping packet is received from a neighbor, the remote time is
compared against the local time.  If the remote time is "greater than"
the local time, then the local clock is advanced.  Similarly, if the
remote time is "less than" the local time, the local clock is retarded.

Since time rolls over once every VIRTUAL_DAY, the meaning of "greater
than" and "less than" must be considered carefully.  In particular,
define:

        err = MOD(remote-local, VIRTUAL_DAY).

If err is less than VIRTUAL_DAY/2, then the remote time is ahead of
the local time by err units. If err is greater than VIRTUAL_DAY/2,
then local time is ahead by (VIRTUAL_DAY-err) units.  Note that if
err exactly equals VIRTUAL_DAY/2, then it's undefined as to which is
ahead.

The algorithm here is designed to set local time to an equally
weighted average between remote time and local time, so if err is
less than VIRTUAL_DAY/2, local time is advanced by err/2.  If err
is greater than VIRTUAL_DAY/2, then local time will be retarded by
(VIRTUAL_DAY-err)/2.  Note that the division by two introduces a
roundoff error in the integer arithmetic used here; the low-order
"half bit" is simulated by a random dither.

*/


#define MAX(a, b)       (((a)>(b))?(a):(b))
#define IS_ODD(a)       (((a)&1) != 0)

// send ping once every four seconds (on average)
#define SYNC_PING_INTERVAL (OS_TICS_PER_SECOND * 5)

static vtime_t sVTOffset;
// The offset added to the real time clock to form the local time

static timePayload sTimeStats;
// place to log statistics of the sync mechanism.


void sync_reset() {
  memset(&sTimeStats, 0, sizeof(timePayload));
}

// ============================================================
// real time clock manipulation

vtime_t sync_getLocalTime() {
  // fetch the current time from the RTC (plus offset)
  unsigned char sec, hthsec;
  vtime_t vt;

  do {
    sec = SEC;
    hthsec = HTHSEC;
  } while (sec != SEC);
  vt = sec;
  vt *= 128;
  vt += hthsec + sVTOffset;
  return vt % VIRTUAL_DAY;
}

static void rtc_advance(vtime_t units) {
  sVTOffset += units;
  sVTOffset %= VIRTUAL_DAY;
}

static void rtc_retard(vtime_t units) {
  sVTOffset -= units;
  sVTOffset %= VIRTUAL_DAY;
}
```

```
// ================================================================

void sync_serviceSeg(pkt_t xdata *seg) {
  // Call sync_serviceSeg() when a SEG_TYPE_PING packet is received
  // from a neighbor.  The packet has already been time stamped at the
  // mac layer with the time of arrival.
  pingPayload xdata *pp = pkt_payload(seg);
  vtime_t err;
  bit advance;

  // note another ping packet received
  sTimeStats.fSyncRcvd++;

  // Careful handling of unsigned numbers, mod VIRTUAL_DAY
  // Pretend VIRTUAL_DAY is 60 seconds (one minute):
  // case A: x=15 r=05, e=10 => advance by e/2
  // case B: x=55 r=05, e=50 => retard by (60-e)/2
  // case C: x=05 r=15, e=10 => retard by e/2
  // case D: x=05 r=55, e=50 => advance by (60-e)/2

  if (pp->fTimeX > pp->fTimeR) {
    err = pp->fTimeX - pp->fTimeR;
    advance = 1;        // assume case A

  } else if (pp->fTimeX < pp->fTimeR) {
    err = pp->fTimeR - pp->fTimeX;
    advance = 0;        // assume case C

  }

  if (err > VIRTUAL_DAY/2) {
    err = VIRTUAL_DAY - err;
    advance = !advance;
  }

  // sTimeStats.fMaxErr is a "leaky peak detector"
  // Note that with this code, fMaxErr will "stick" at (2^2)-1, or 3.
  // I could use fixpoint arithmetic to make it better, but it's not
  // crucial.
  sTimeStats.fMaxErr -= (sTimeStats.fMaxErr >> 2);   // slow decay...
  sTimeStats.fMaxErr = MAX(sTimeStats.fMaxErr, err); // ... fast rise


  if (err != 0) {
    if (IS_ODD(err)) err += rand() & 1; // dither before divide by 2
    err = err/2;                        // divide by 2

    if (advance) {
      SCREEN_TASK(("tx=%4u tr=%4u er=+%u, mx=%u",
        pp->fTimeX, pp->fTimeR, err, sTimeStats.fMaxErr));
      rtc_advance(err);
    } else {
      SCREEN_TASK(("tx=%4u tr=%4u er=-%u, mx=%u",
        pp->fTimeX, pp->fTimeR, err, sTimeStats.fMaxErr));
      rtc_retard(err);
    }
  } else {
      SCREEN_TASK(("tx=%4u tr=%4u er=0, mx=%u",
        pp->fTimeX, pp->fTimeR, sTimeStats.fMaxErr));
  }
}

// ================================================
// Sync thread.
// Send a periodic PING message to immediate neighbors
// once every PING_SECONDS seconds.

static void _bide(unsigned int tics) {
  // wait for the given number of tics to elapse.  Each tic is
  // approx 9.6 mSec, or 106 tics per second.
  // SCREEN_TASK(("_bide(1) %x", tics));
  while (tics != 0) {
    unsigned char t;
    t = (tics > UCHAR_MAX)?UCHAR_MAX:tics;
    os_wait2(K_TMO, t);
    tics -= t;
  }
}

void sync_task(void) _task_ SYNC_TASK {

  // one-time initialization
  sVTOffset = 0;
  sync_reset();

  while (1) {
    // initiate a ping packet to advertize this node's local time
    pkt_t xdata *pkt = pkt_alloc();
    pkt_type(pkt) = SEG_TYPE_PING;
    pkt_size(pkt) =  sizeof(pingPayload);

    SCREEN_TASK(("sync_task(1) %x", sync_getLocalTime()));
    // BIG NOTE: Since the packet may spend an unknown amount of time
    // in the MAC queue, the pingPayload->fTimeX field is filled in by
    // rad.c just prior to transmission.  This reduces timing errors.
    //
    // Lesser note: If a node packet decides to relay a ping packet
    // rather than originate it, it would be a problem if the packet
    // went out with the orinator's nodeID.  Consequently, the fNodeID
    // field is filled in at the same time as the fTimeX field to
    // prevent this possibility.
    mac_xmitPkt(pkt);

    // note another ping transmitted
    sTimeStats.fSyncSent++;

    // sleep for SYNC_PING_INTERVAL +/- 50%
    _bide((SYNC_PING_INTERVAL/2) + (rand() % SYNC_PING_INTERVAL));
  }
}

pkt_t xdata *sync_report(pkt_t xdata *next) {
  // create a packet that reports the current time statistics for this
  // node.  Copy the payload from the local sTimeStats.
  pkt_t xdata *pkt = pkt_alloc();
  timePayload xdata *tp = pkt_payload(pkt);
```

```
  pkt_type(pkt) = SEG_TYPE_TIME;
  pkt_size(pkt) = sizeof(timePayload);
  pkt_next(pkt) = next;
  sTimeStats.fLocalTime = sync_getLocalTime();
  memcpy(pkt_payload(pkt), &sTimeStats, sizeof(timePayload));
  return pkt;
}
```

# File: vector.h

```
#ifndef VECTOR_H
#define VECTOR_H
// -*- Mode: C++ -*-
//
// File:        vector.h
// Description: manage a fixed sized array of pointer-sized objects
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// A vector is a sequence of pointer-sized elements, densely packed
// starting at index 0.  The objects referred to by the vector
// are always in OBJECT_SPACE, which you can redefine
// according to taste.
#define OBJECT_SPACE xdata

// obj_t is a general pointer into OBJECT_SPACE
typedef void OBJECT_SPACE * obj_t;

typedef struct _vector_t {
  unsigned char fCount;         // number of elements in the array
  unsigned char fCapacity;      // size of the array
  obj_t fElements[1];           // a dense array of objects
} vector_t;

// create static storage for a vector, masquerading as an array of char
#define DEFINE_VECTOR(name, capacity) \
    char name[sizeof(vector_t) + ((capacity - 1) * sizeof(obj_t))]
```

```
// turn a static char reference into a vector pointer.
#define VECTOR(v) ((vector_t *)&v)

// ============================================================

// initialize a vector.  Must call this before first use
// void vector_init(vector_t *v, unsigned char capacity)
#define vector_init(v, capacity)     \
    (v)->fCount = 0; (v)->fCapacity = (capacity)

void vector_print(vector_t *v);

vector_t *vector_insert(vector_t *v, obj_t elem, unsigned char index);
// insert element into the vector.  return v if inserted, or null if
// the vector was full prior to the call or if index is out of range.

obj_t vector_remove(vector_t  *v, unsigned char index);
// remove and return the element at the given index, or return
// null if index is out of range.

vector_t *vector_swap(vector_t *v, unsigned char i1, unsigned char i2);
// swap two elements in the vector.  return null if i1 or i2 are out
// of range.

obj_t vector_shove(vector_t *v, obj_t element);
// Like vector_push(), inserts element at the high end of the
// array.  Unlike vector_push(), removes the first element and
// returns it to make room for the new element as needed.

unsigned char vector_index_of(vector_t *v, obj_t element);
// return the index of the element in the vector, or -1 if not found

obj_t vector_ref(vector_t *v, unsigned char index);
// return the indexth entry of the table, or null if index out of range

vector_t *vector_set(vector_t *v, obj_t element, unsigned char index);
// set the indexth entry of the table to element.  Returns v on
// success, null if index is out of range.

// ============================================================
// Everything else are macro definitions...

// vector_t *vector_clear(vector_t *v);
#define vector_clear(v) ((v)->fCount) = 0, (v)

// unsigned char vector_count(vector_t *v);
#define vector_count(v) ((v)->fCount)

// unsigned char vector_capacity(vector_t *v);
#define vector_capacity(v) ((v)->fCapacity)

// obj_t *vector_elements(vector_t *v);
#define vector_elements(v) ((v)->fElements)

// vector_t *vector_push(vector_t *v, obj_t element);
#define vector_push(v, e) vector_insert((v), (e), (v)->fCount)

// obj_t vector_pop(vector_t *v);
#define vector_pop(v) vector_remove((v), ((v)->fCount)-1)

// vector_t vector_enqueue(vector_t *v, obj_t element);
#define vector_enqueue(v, e) vector_insert((v), (e), (v)->fCount)

// obj_t vector_dequeue(vector_t *v);
#define vector_dequeue(v) vector_remove((v), 0)

// boolean vector_is_empty(vector_t *v);
#define vector_is_empty(v) ((v)->fCount == 0)

// boolean vector_is_full(vector_t *v);
#define vector_is_full(v) ((v)->fCount == (v)->fCapacity)

// ============================================================
// fast versions - call only when you know arguments to be safe!

void fast_vector_insert(vector_t *v, obj_t element, unsigned char index);

obj_t fast_vector_remove(vector_t *v, unsigned char index);

void fast_vector_swap(vector_t *v, unsigned char index1, unsigned char
index2);

// obj_t fast_vector_ref(vector_t *v, unsigned char index);
#define fast_vector_ref(v, i) ((v)->fElements[(i)])

// void fast_vector_set(vector_t *v, obj_t element, unsigned char index);
#define fast_vector_set(v, e, i) ((v)->fElements[(i)]) = e

#define fast_vector_push(v, e) fast_vector_insert((v), (e), (v)->fCount)

#define fast_vector_pop(v) vector_remove((v), ((v)->fCount)-1)

#define fast_vector_enqueue(v, e) fast_vector_insert((v), (e), (v)->fCount)

#define fast_vector_dequeue(v) fast_vector_remove((v), 0)

#endif
```

# File: vector.c

```
// -*- Mode: C++ -*-
//
// File:        vector.c
// Description: manage fixed size arrays of pointer sized objects
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
```

```
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "vector.h"
#include <stdio.h>

#ifndef NULL
#define NULL (void *)0
#endif

#ifdef UNCALLED_SEGMENT
void vector_print(vector_t *v) {
  unsigned char i;
  printf("\r\nv<%p>, cap=%bd, count=%bd ", v, v->fCapacity, v->fCount);
  for (i=0; i<v->fCount; i++) {
    printf("[%d]%p%s",i,(v->fElements)[i], (i==v->fCount-1)?"":", ");
  }
}
#endif

#ifdef UNCALLED_SEGMENT
vector_t *vector_insert(vector_t *v, obj_t elem, unsigned char index) {
  if ((index < 0)||(index > v->fCount)||(v->fCount >= v->fCapacity)) {
    return NULL;
  }
  fast_vector_insert(v, elem, index);
  return v;
}
#endif

void fast_vector_insert(vector_t *v, obj_t elem, unsigned char index) {
  obj_t *e0, *e1, *e2;

  // Open a slot for an element at index by rippling all higher
  // elements up by one.
  e0 = &(v->fElements[index]);
  e2 = &(v->fElements[v->fCount++]);
  e1 = e2 - 1;
  while (e2 > e0) {
    *e2-- = *e1--;
  }
  *e0 = elem;
}

obj_t vector_remove(vector_t *v, unsigned char index) {
  if ((index < 0) || (index >= v->fCount)) {
    return NULL;
  }
  return fast_vector_remove(v, index);
}

obj_t fast_vector_remove(vector_t *v, unsigned char index) {
  obj_t *e1, *e2, elem;

  e1 = &(v->fElements[index]);
  e2 = e1+1;
  elem = *e1;

  // close the slot at index by rippling elements down (towards [0])
  // by one.
  v->fCount--;
  while (index++ < v->fCount) {
    *e1++ = *e2++;
  }
  return elem;
}

obj_t vector_shove(vector_t *v, obj_t element) {
  // Like vector_push(), but removes (and returns) the first element
  // if the vector was full, making room for the new element.  Good
  // for LRU structures.
  obj_t shoved = NULL;

  if (vector_is_full(v)) {
    shoved = fast_vector_remove(v, 0);
  }
  fast_vector_insert(v, element, v->fCount);
  return shoved;
}


#ifdef UNCALLED_SEGMENT
vector_t *vector_swap(vector_t *v, unsigned char i1, unsigned char i2) {
  obj_t tmp, *elems;
  if ((i1 < 0) || (i1 >= v->fCount) ||
      (i2 < 0) || (i2 >= v->fCount)) {
    return NULL;
  }
  elems = v->fElements;
  tmp = elems[i1];
  elems[i1] = elems[i2];
  elems[i2] = tmp;
  return v;
}
#endif

#ifdef UNCALLED_SEGMENT
void fast_vector_swap(vector_t *v, unsigned char i1, unsigned char i2) {
```

```
  obj_t *elems, temp;

  elems = v->fElements;
  temp = elems[i1];
  elems[i1] = elems[i2];
  elems[i2] = temp;
}
#endif

#ifdef UNCALLED_SEGMENT
unsigned char vector_index_of(vector_t *v, obj_t elem) {
  // return the index of the element in the vector, or -1 if not found
  obj_t *elems = v->fElements;
  unsigned char i = v->fCount;

  while (--i > 0) {
    if (elems[i] == elem) return i;
  }
  return -1;
}
#endif

#ifdef UNCALLED_SEGMENT
obj_t vector_ref(vector_t *v, unsigned char index) {
  if ((index < 0) || (index >= v->fCount)) {
    return NULL;
  } else {
    return fast_vector_ref(v, index);
  }
}
#endif

#ifdef UNCALLED_SEGMENT
vector_t *vector_set(vector_t *v, obj_t elem, unsigned char index) {
  if ((index < 0) || (index >= v->fCount)) {
    return NULL;
  }
  fast_vector_set(v, elem, index);
  return v;
}
#endif


// ================================================================
// ================================================================
// test suite for vector code
// #define TEST_VECTOR
#ifdef TEST_VECTOR

#include "arbor.h"
#include <stdio.h>

#define true (1==1)
#define false (1==0)

void print_vector(vector_t *v) {
  unsigned char i;
  printf("v<%p>, count=%d, ", v, v->fCount);
  for (i=0; i<v->fCount; i++) {
    printf("[%d]%p%s",i,(v->fElements)[i],
           (i==v->fCount-1)?"\n":", ");
  }
}

void test(obj_t got, obj_t expected, char *msg) {
  if (got != expected) {
    printf("%s: got %p, expected %p\n", msg, got, expected);
  } else {
    printf("%s: okay\n", msg);
  }
}

#define TEST(got, exp, s) test(((obj_t)(got)), ((obj_t)(exp)), s)

#define CAPACITY 6

DEFINE_VECTOR(v, CAPACITY);

void init() _task_ MAIN_TASK {

  PLLCON = 0x00; // 12 MHz
  LED_INIT();
  LED_ON(AMBER_LED);
  //ALL_LEDS_ON();
  serial_init();    // set up baud rate

  os_wait2(K_TMO, 4);
  puts("\r\n\r\nvector test\r\n");

  vector_init(VECTOR(v), CAPACITY);

  TEST(vector_dequeue((vector_t *)&v), NULL, "getting from new vector");
  TEST(vector_enqueue(VECTOR(v), (obj_t)0x1), VECTOR(v), "vector_enqueue
returns vector");
  vector_enqueue(VECTOR(v), (obj_t)0x2);
  vector_enqueue(VECTOR(v), (obj_t)0x3);
  vector_enqueue(VECTOR(v), (obj_t)0x4);
  TEST(vector_count(VECTOR(v)), 4, "vector size mismatch");
  TEST(vector_dequeue(VECTOR(v)), 0x1, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x2, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x3, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x4, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), NULL, "getting from empty vector");

  TEST(vector_is_empty(VECTOR(v)), true, "vector is empty");
  // TEST(vector_is_full(VECTOR(v)), false, "vector is not full");

  vector_enqueue(VECTOR(v), (obj_t)0x22);
  vector_enqueue(VECTOR(v), (obj_t)0x33);
  vector_enqueue(VECTOR(v), (obj_t)0x44);
  vector_enqueue(VECTOR(v), (obj_t)0x55);
  vector_enqueue(VECTOR(v), (obj_t)0x66);
  vector_enqueue(VECTOR(v), (obj_t)0x77);
  TEST(vector_is_empty(VECTOR(v)), false, "vector is not empty");
  TEST(vector_is_full(VECTOR(v)), true, "vector is full");
  TEST(vector_enqueue(VECTOR(v), (obj_t)0x88), NULL, "putting to full
```

```
vector");
  TEST(vector_count(VECTOR(v)), CAPACITY, "full count = capacity");

  TEST(vector_remove(VECTOR(v), vector_index_of(VECTOR(v), (obj_t)0x55)),
       0x55,
       "remove from middle");
  TEST(vector_count(VECTOR(v)), 5, "count after remove");

  TEST(vector_remove(VECTOR(v), vector_index_of(VECTOR(v),
(obj_t)"bogus")),
       NULL,
       "remove bogus");
  TEST(vector_count(VECTOR(v)), 5, "count after remove bogus");

  TEST(vector_dequeue(VECTOR(v)), 0x22, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x33, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x44, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x66, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x77, "getting from vector");

  TEST(vector_count(VECTOR(v)), 0, "count = 0");

  TEST(vector_clear(VECTOR(v)), VECTOR(v), "clear vector");
  vector_enqueue(VECTOR(v), (obj_t)0x111);
  vector_enqueue(VECTOR(v), (obj_t)0x333);
  vector_enqueue(VECTOR(v), (obj_t)0x444);
  print_vector(VECTOR(v));
  TEST(vector_insert(VECTOR(v), (obj_t)0x222, 1), VECTOR(v), "insert at
1");
  print_vector(VECTOR(v));
  TEST(vector_dequeue(VECTOR(v)), 0x111, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x222, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x333, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), 0x444, "getting from vector");
  TEST(vector_dequeue(VECTOR(v)), NULL, "getting from empty vector");

  TEST(vector_insert(VECTOR(v), (obj_t)0x111, 1), NULL, "insert beyond
end");
  TEST(vector_insert(VECTOR(v), (obj_t)0x111, 0), VECTOR(v), "insert at
0");
  print_vector(VECTOR(v));
  TEST(vector_insert(VECTOR(v), (obj_t)0x222, 1), VECTOR(v), "insert at
1");
  print_vector(VECTOR(v));
  TEST(vector_insert(VECTOR(v), (obj_t)0x999, 0), VECTOR(v), "insert at
0");
  print_vector(VECTOR(v));

  TEST(vector_insert(VECTOR(v), (obj_t)0x888, 3), VECTOR(v), "insert at
end");
  // v = 0x999 0x111 0x222 0x888
  TEST(vector_ref(VECTOR(v), 4), NULL, "ref beyond end");
  TEST(vector_ref(VECTOR(v), -1), NULL, "ref before beginning");
  TEST(vector_ref(VECTOR(v), 2), 0x222, "ref 2");
  TEST(vector_set(VECTOR(v), (obj_t)0x333, 4), NULL, "set beyond end");
  TEST(vector_set(VECTOR(v), (obj_t)0x333, 2), VECTOR(v), "set 2");
  TEST(vector_ref(VECTOR(v), 2), 0x333, "ref 2 redux");
  TEST(vector_swap(VECTOR(v), 1, 2), VECTOR(v), "swap");
  TEST(vector_ref(VECTOR(v), 2), 0x111, "ref 2 post swap");
  TEST(vector_swap(VECTOR(v), 1, 2), VECTOR(v), "swap");
  TEST(vector_ref(VECTOR(v), 2), 0x333, "ref 2 post swap");

  // v = 0x999 0x111 0x333 0x888
  TEST(vector_remove(VECTOR(v), 4), NULL, "remove beyond end");
  TEST(vector_remove(VECTOR(v), 2), 0x333, "remove middle");
  TEST(vector_remove(VECTOR(v), 0), 0x999, "remove first");
  TEST(vector_remove(VECTOR(v), 1), 0x888, "remove last");
  TEST(vector_remove(VECTOR(v), 0), 0x111, "remove first and last");
}

#endif
```

# APPENDIX C *ArborNet "BART" Code Listing*

Following is the ArborNet C source code that is executed by the PIC16F84 "BART" radio processor on the Constellation board. More information on the ArborNet system can be found in Chapter 7.

## File: bart.h

```
#ifdef _BART_H
#nolist
#else
#define _BART_H
// -*- Mode: C++ -*-
//
// File:        bart.h
// Description: general system definitions for BART radio chip
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// ============================================================
// I/O pin definitions

struct PORT_A_MAP {
  int HDATA_LO:4;                       // A0:3 low nibble of host data
  boolean RADIO_RCV;                    // A4 serial data (from radio)
  boolean unused_a5;
  boolean unused_a6;
  boolean unused_a7;
} PORT_A;
#byte PORT_A = 5

struct PORT_B_MAP {
  int HDATA_HI:4;                       // B0:3 high nibble of host data
  boolean RADIO_XMT;                    // B4 serial data (to radio)
  boolean BART_READY;                   // B5 handshake (to host)
  boolean RCV_MODE;                     // B6 recv/xmit control (from host)
  boolean HOST_CLK;                     // B7 handshake (from host)
} PORT_B;
#byte PORT_B = 6

// TRIS bits for transmit mode (from host to PIC to radio)
struct PORT_A_MAP const PORT_A_XMT = {0xf, 1, 0, 0, 0};
struct PORT_B_MAP const PORT_B_XMT = {0xf, 0, 0, 1, 1};

// TRIS bits for receive mode (from radio to PIC to host)
struct PORT_A_MAP const PORT_A_RCV = {0x0, 1, 0, 0, 0};
struct PORT_B_MAP const PORT_B_RCV = {0x0, 0, 0, 1, 1};

// ============================================================
// BART_READY is a low true signal

#define ASSERT_READY(b)         PORT_B.BART_READY = !(b)
#define READY_IS_ASSERTED()     (!PORT_B.BART_READY)


// ============================================================
// Timing definitions

// The bit period for serial (radio) data, measured in TMR0 tics
// With a 20 MHz crystal, one tic is .2 uSec, so the bit period
// is 8.8 uSec, or 113.63 KBaud
#define BIT_PERIOD      44
```

```
#endif
#list
```

# File: bart.c

```
// -*- Mode: C++ -*-
//
// File:        bart.c
// Description: initialization and main loop
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#case
#include <16F84.H>

// run with watchdog timer ON so PIC will restart if it gets hung.
#fuses HS,WDT,NOPROTECT,PUT

// All timing is done with TMR0, so software delays aren't needed.
// #use DELAY(clock=20000000)

#use fast_io(A)
#use fast_io(B)

typedef short int boolean;

// if defined, put state vars on parallel port
// #define BLAT_STATE

#include "procregs.h"
#include "utils.h"
#include "bart.h"

#include "codec.h"
#include "fifo.h"
#include "host.h"
#include "radio.h"
#include "sync.h"

// ================================================================
// globals

// gXmtActive is true as long as we're in transmit mode (RCV_MODE=0)
// and there are more bits to be sent in the fifo.  It will be set
// to false after the last bit has ben transmitted.
short int gXmtActive;

// keep track of how many times reset has been called.  Assumes
// memory is not zeroed at reset.
//
int gResetCount;

// ================================================================
// included files
//
// Defines encode_nibble() and decode_nibble()

// conversion between 4 bit decoded and 6 bit dc-balanced encoded
#include "codec.c"

// a simple FIFO for buffering data between host and radio
#include "fifo.c"

// managing parallel I/O with the host
#include "host.c"

// managing serial I/O with the radio
#include "radio.c"

// establishing sync between transmitters and receivers
#include "sync.c"

// ================================================================
// initialization and main code

// set up for transmit mode
#inline
void setup_transmit() {
  set_tris_a(PORT_A_XMT);
  set_tris_b(PORT_B_XMT);
  HOST_SETUP_TX();
  RADIO_SETUP_TX();
  FIFO_RESET();
}

// set up for receive mode
```

```
#inline
void setup_receive() {
  set_tris_a(PORT_A_RCV);
  set_tris_b(PORT_B_RCV);
  HOST_SETUP_RX();
  RADIO_SETUP_RX();
  FIFO_RESET();
}

// ================================================================
// blat_state() - put state vars on parallel port for debugging

#ifndef BLAT_STATE

#define blat_state()    /* nop */

#else

#inline
void blat_state() {
  output_high(PIN_B3);
#asm
  // radio state on low nibble
  movf  gRState, w
  xorwf PORT_A, w
  andlw 0x0f
  xorwf PORT_A, f

  // host state on high nibble
  movf gHState, w
  xorwf PORT_B, w
  andlw 0x07
  xorwf PORT_B, f

#endasm
  output_low(PIN_B3);
}

#endif


// ================================================================
// ================================================================
// main()

void main() {
  int i;

  set_tris_a(PORT_A_XMT);
  set_tris_b(PORT_B_XMT);

  // this loop makes it obvious to a 'scope that we've reset.
  i = 30;
  do {
    ASSERT_READY(1);
    delay_cycles(43);
    ASSERT_READY(0);
    delay_cycles(43);
  } while (--i);
  gResetCount++;

  setup_counters(RTCC_INTERNAL, WDT_18MS);

  while (1) {

    // enter transmit mode, stay there until the host asserts RCV_MODE
    // and fifo has finished transmitting its contents
    setup_transmit();
    gXmtActive = 1;
    do {
      service_radio_xmt();              // send serial bits to radio
      restart_wdt();
    } while (gXmtActive);

    // enter receive mode, stay there until the host drops RCV_MODE
    setup_receive();
    do {
      service_radio_rcv();              // receive serial bits from radio
      restart_wdt();
    } while (PORT_B.RCV_MODE);
  }
}
```

## File: codec.h

```
#ifdef _CODEC_H
#nolist
#else
#define _CODEC_H
// -*- Mode: C++ -*-
//
// File:        codec.h
// Description: header file for encoding / decoding DC balanced nibbles
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// two special values returned by codec_decode()

#define DECODE_ILLEGAL 0x80
#define DECODE_SYNCH  0xff

// Convert binary nibble (in low order 4 bits of nibble) into 6 bit,
// DC balanced values.  Values may be received using the W_RECV()
// macro, as follows:
//   codec_encode(nibble);
//   W_RECV(result);
//
int codec_encode(int nibble);

// Convert DC balanced "hexlet" (in low order 6 bits of hexlet) into
// four bit value (returned in W register).  Returns DECODE_ILLEGAL
// for illegal 6 bit patterns, returns DECODE_SYNCH if a sync header
// pattern is given.
//
// Values may be fetched using the W_RECV() macro, as in:
//    codec_decode(hexlet);
//    W_RECV(result);
//    if (result == DECODE_ILLEGAL) error();
//
int codec_decode(int hexlet);

#endif
#list
```

## File: codec.c

```
// -*- Mode: C++ -*-
//
// File:        codec.c
// Description: Convert between 4 bit and 6 bit DC-balanced values
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "utils.h"

// TWO BIG WARNINGS:
// [1] Make sure neither dispatch table crosses a page boundary -
//     if so, you must use DISPATCH() rather than SHORT_DISPATCH().
// [2] Make sure that codec_{de en}code is called AS A SUBROUTINE, via
//     CALL rather than GOTO.  (The CCS compiler will use CALL whenever
//     there are two or more subroutine calls in the program.)
```

```
// Encode 4 bit nibble as a 6 bit DC-balanced value.  Upon return, the
// encoded 6 bits are "left justified" in the W register and must be
// stored somewhere via the W_RECV() macro (see utils.h).  Example call:
//   encode_nibble(decoded);
//   W_RECV(result);
// The reason for left justification is that we will be shifting out
// the bits MSB first.  "prejustifying" them saves an extra shift
// operation.

int codec_encode(int nibble) {
  nibble &= 0x0f;               // mask to 4 bits
#asm
  SHORT_DISPATCH(nibble)
    retlw 0b01010100 // 0000
    retlw 0b11000100 // 0001
    retlw 0b11001000 // 0010
    retlw 0b10001100 // 0011
    retlw 0b11010000 // 0100
    retlw 0b10010100 // 0101
    retlw 0b10011000 // 0110
    retlw 0b01011000 // 0111
    retlw 0b01101000 // 1000
    retlw 0b10100100 // 1001
    retlw 0b10101000 // 1010
    retlw 0b00101100 // 1011
    retlw 0b10110000 // 1100
    retlw 0b00110100 // 1101
    retlw 0b00111000 // 1110
    retlw 0b01110000 // 1111
#endasm
}

// Convert DC balanced "hexlet" (in low order 6 bits of hexlet) into
// four bit value (returned in W register).  Returns DECODE_ILLEGAL
// for illegal 6 bit patterns, returns DECODE_SYNCH if a sync header
// pattern is given.
//
// Values may be fetched using the W_RECV() macro, as in:
//    codec_decode(hexlet);
//    W_RECV(result);
//    if (result == DECODE_ILLEGAL) error();
//
int codec_decode(int hexlet) {
  hexlet &= 0x3f;               // mask to 6 bits
#asm
  SHORT_DISPATCH(hexlet)
    retlw DECODE_ILLEGAL        // 000000 not used
    retlw DECODE_ILLEGAL        // 000001 not used
    retlw DECODE_ILLEGAL        // 000010 not used
    retlw DECODE_ILLEGAL        // 000011 not used
    retlw DECODE_ILLEGAL        // 000100 not used
    retlw DECODE_ILLEGAL        // 000101 not used
    retlw DECODE_ILLEGAL        // 000110 not used
    retlw DECODE_ILLEGAL        // 000111 "anti synch header"
    retlw DECODE_ILLEGAL        // 001000 not used
    retlw DECODE_ILLEGAL        // 001001 not used
    retlw DECODE_ILLEGAL        // 001010 not used
    retlw 0b00001011            // 001011
    retlw DECODE_ILLEGAL        // 001100 not used
    retlw 0b00001101            // 001101
    retlw 0b00001110            // 001110
    retlw DECODE_ILLEGAL        // 001111 not used
    retlw DECODE_ILLEGAL        // 010000 not used
    retlw DECODE_ILLEGAL        // 010001 not used
    retlw DECODE_ILLEGAL        // 010010 not used
    retlw DECODE_ILLEGAL        // 010011 not used
    retlw DECODE_ILLEGAL        // 010100 not used
    retlw 0b00000000            // 010101
    retlw 0b00000111            // 010110
    retlw DECODE_ILLEGAL        // 010111 not used
    retlw DECODE_ILLEGAL        // 011000 not used
    retlw DECODE_ILLEGAL        // 011001 not used
    retlw 0b00001000            // 011010
    retlw DECODE_ILLEGAL        // 011011 not used
    retlw 0b00001111            // 011100
    retlw DECODE_ILLEGAL        // 011101 not used
    retlw DECODE_ILLEGAL        // 011110 not used
    retlw DECODE_ILLEGAL        // 011111 not used
    retlw DECODE_ILLEGAL        // 100000 not used
    retlw DECODE_ILLEGAL        // 100001 not used
    retlw DECODE_ILLEGAL        // 100010 not used
    retlw 0b00000011            // 100011
    retlw DECODE_ILLEGAL        // 100100 not used
    retlw 0b00000101            // 100101
    retlw 0b00000110            // 100110
    retlw DECODE_ILLEGAL        // 100111 not used
    retlw DECODE_ILLEGAL        // 101000 not used
    retlw 0b00001001            // 101001
    retlw 0b00001010            // 101010
    retlw DECODE_ILLEGAL        // 101011 not used
    retlw 0b00001100            // 101100
    retlw DECODE_ILLEGAL        // 101101 not used
    retlw DECODE_ILLEGAL        // 101110 not used
    retlw DECODE_ILLEGAL        // 101111 not used
    retlw DECODE_ILLEGAL        // 110000 not used
    retlw 0b00000001            // 110001
    retlw 0b00000010            // 110010
    retlw DECODE_ILLEGAL        // 110011 not used
    retlw 0b00000100            // 110100
    retlw DECODE_ILLEGAL        // 110101 not used
    retlw DECODE_ILLEGAL        // 110110 not used
    retlw DECODE_ILLEGAL        // 110111 not used
    retlw DECODE_SYNCH          // 111000 synch pattern
    retlw DECODE_ILLEGAL        // 111001 not used
    retlw DECODE_ILLEGAL        // 111010 not used
    retlw DECODE_ILLEGAL        // 111011 not used
    retlw DECODE_ILLEGAL        // 111100 not used
    retlw DECODE_ILLEGAL        // 111101 not used
    retlw DECODE_ILLEGAL        // 111110 not used
    retlw DECODE_ILLEGAL        // 111111 not used
#endasm
}
```

## File: fifo.h

```
#ifdef _FIFO_H
#nolist
#else
#define _FIFO_H
// -*- Mode: C++ -*-
//
// File:        fifo.h
// Description: header file for fifo routines
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// fifo capacity must be a power of 2!!
#define FIFO_CAPACITY   32
#define FIFO_MASK       (FIFO_CAPACITY-1)

#define FIFO_IS_EMPTY() (fifoLen == 0)
#define FIFO_IS_FULL()  (fifoLen == FIFO_CAPACITY)

#define FIFO_RESET()    fifoPut=0; fifoGet=0; fifoLen=0

// Store a byte in the fifo.  Assumes that the caller has previously
// checked for overflow, as in !FIFO_IS_FULL().
//
#inline
void fifo_put(int &b);

// Fetch a byte from the fifo.  Assumes that the caller has previously
// checked for underflow, as in !FIFO_IS_EMPTY().  Returns value in W,
// and must be fetched as in W_RECV(val);
//
#inline
void fifo_get();

#endif
#list
```

## File: fifo.c

```
// -*- Mode: C++ -*-
//
// File:        fifo.c
// Description: very efficient and dangerous fifo routines
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.
#include "fifo.h"
#include "procregs.h"

int fifoPut;
int fifoGet;
int fifoLen;                            // # of byte stored in fifo
int fifo[FIFO_CAPACITY];

// ============================================================
// FIFO code

// store a byte in fifo.  Assumes caller has checked for overflow
```

```
#inline
void fifo_put(int &b) {
  // if (fifoLen == FIFO_CAPACITY) return;      // overflow,  sorry.
  fifoLen++;
#ifdef DONT_BUM_CYCLES
  fifo[fifoPut++] = b;
#else
#asm
  movf   fifoPut,W
  incf   fifoPut,F
  addlw  fifo
  movwf  FSR
  movf   b,W
  movwf  INDF
#endasm
#endif
  fifoPut &= FIFO_MASK;
}

// fetch a byte from fifo.  Assumes caller has checked for underflow.
// Returns value in W register.
#inline
void fifo_get() {
  fifoLen--;
#asm
  movf   fifoGet,W
  incf   fifoGet,F
  addlw  fifo
  movwf  FSR            // p = &fifo[fifoGet++]
  movlw  FIFO_MASK      // fifoGet &= FIFO_MASK
  andwf  fifoGet,F      // ...
  movf   INDF,W         // w = *p
#endasm
#endif
}
```

# File: host.h

```
#ifdef _HOST_H
#nolist
#else
#define _HOST_H
// -*- Mode: C++ -*-
//
// File:        host.h
// Description: service host parallel port
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// warm reset code

#define HOST_SETUP_TX()                 \
        gHState = SHX_A;                \
        ASSERT_READY(0)

#define HOST_SETUP_RX()                 \
        gHState = SHR_A;                \
        ASSERT_READY(0)

#inline
void service_host_rcv();

void service_host_xmt();

#endif
#list
```

# File: host.c

```
// -*- Mode: C++ -*-
//
// File:        host.c
// Description: manage communication between BART and host processor
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.
/
// This file implements the following:
//
// void service_host_rcv();
// In receive mode (reading data from radio), monitor the software
// FIFO for bytes and send them to the host via the parallel port
// as they become available.
//
// void service_host_xmt();
// In transmit mode (sending data to radio), monitor the parallel
// port for data transfers from the host and store incoming bytes
// in the software fifo as they are sent.
//
// Timing requirements:
// Both service_host_rcv() and service_host_xmt() share processor
// cycles with the software UART defined in radio.c.  Consequently,
// these routines must execute and return very quickly so as not to
// perturb the serial timing.  As a rule of thumb, these functions
// should consume only a small fraction of BIT_PERIOD tics.
//
// Edit History:
//
// 08 Oct 2000: r@media.mit.edu
// service_host_xmt() was getting stuck in stateD.  fixed typo
//
// End of Edit History

#include "bart.h"
#include "procregs.h"
#include "utils.h"
#include "host.h"

// ================================================================

// Variables can be safely shared between service_host_rcv() and
// service_host_xmt() since the system can't be receiving and
// transmitting at the same time.

int gHState;
int gHByteBuf;

// ================================================================
// service_host_rcv()
//
// Called regularly when the radio is in receive mode, this routine
// transfers data from the FIFO to the parallel port.  The caller
// requires 25 cycles, leaving 44-25 = 19 cycles for this routine.
// The odd ordering of the clauses (putting shr_B first after the
// dispatch) shaves off two critical cycles.
//
#inline
void service_host_rcv() {
#asm
  SHORT_DISPATCH(gHState)
#define SHR_C 0
    goto shr_C          // post data to port, await !HOST_CLOCK
#define SHR_A 1
    goto shr_A          // bart_ready <= fifo_state, advance if avail
    // goto shr_B       // wait for HOST_CLOCK, pull data from FIFO
    // v === fall through === v
#endasm

 shr_B:
  // Await HOST_CLOCK, then fetch a byte from the fifo
  if (PORT_B.HOST_CLK) {
    fifo_get();
    W_RECV(gHByteBuf);          // fifo_get returns value in W
    gHState = SHR_C;    // gHState was 2(B), now 0(C)
  }
  return;

 shr_A:
  // Wait for a byte to become available in the fifo.
  // Announce readiness in BART_READY.
  if (FIFO_IS_EMPTY()) {
    ASSERT_READY(0);
  } else {
```

```
    ASSERT_READY(1);
    gHState++;          // gHState was 1(A), now 2(B)
  }
  return;

 shr_C:                 // gHState = 0
  // Output the byte to the host port, await !HOST_CLK
#ifdef DONT_BUM_CYCLES
  PORT_A.HDATA_LO = gHByteBuf;
  swap(gHByteBuf);
  PORT_B.HDATA_HI = gHByteBuf;
#else
#ifndef BLAT_STATE
#asm
  // This code requires that HDATA_LO and HDATA_HI appear at the
  // low four bits of PORT_A and PORT_B respectively.
  // movf  gHByteBuf, w
  // xorwf PORT_A, w
  // andlw 0x0f
  // xorwf PORT_A, f
  // super big cheeze hack: lsnibble of PORT_A is where we put
  // the bits.  hsnibble of PORT_A is either inputs (A4) or
  // unused.  cycle shaving to the max...
  movf  gHByteBuf, w
  movwf PORT_A

  swapf gHByteBuf, w
  xorwf PORT_B, w
  andlw 0x0f
//  andlw 0x07          // ### leave B3 for debugging
  xorwf PORT_B, f
#endasm
#endif // ifndef BLAT_STATE

#endif // ifdef DONT_BUM_CYCLES

  if (!PORT_B.HOST_CLK) {
    gHState++;          // gHState was 0(C), now 1(A)
  }
  return;
}

// ================================================================
// service_host_xmt()
// Handle the host port in transmit mode.  Loop as follows:
// Inform host if there's room in the FIFO by raising BART_RDY
// Await HOST_CLOCK, latch data on parallel port
// Store data in fifo, await !HOST_CLOCK
//
void service_host_xmt() {
#asm
  SHORT_DISPATCH(gHState)
#define SHX_A 0
    goto shx_A                  // if room in fifo, assert ready
    goto shx_B                  // await host clock, latch data
    goto shx_C                  // store in fifo, await !host clock
#endasm

 shx_A:
    if (!FIFO_IS_FULL()) {
      ASSERT_READY(1);
      gHState++;
    } else {
      ASSERT_READY(0);
    }
    return;

 shx_B:
    // wait for host clock to go true before latching data
    // on parallel port
    if (PORT_B.HOST_CLK) {
#ifdef DONT_BUM_CYCLES
      gHByteBuf = PORT_B.HDATA_HI;
      swap(gHByteBuf);
      gHByteBuf |= PORT_A.HDATA_LO;
#else
      // This code requires that HDATA_LO and HDATA_HI appear at the
      // low four bits of PORT_A and PORT_B respectively.
      gHByteBuf = (int *)PORT_B & 0x0f;
      swap(gHByteBuf);
      gHByteBuf |= (int *)PORT_A & 0x0f;
#endif
      gHState++;
    }
    return;

 shx_C:
    // store the byte fetched in the previous state, wait for
    // host clock to drop before going to next state.
    if (!PORT_B.HOST_CLK) {
      fifo_put(gHByteBuf);
      gHState = SHX_A;
    }
    return;
}
```

# File: procregs.h

```
#ifdef _PROCREGS_H
#nolist
#else
#define _PROCREGS_H
// -*- Mode: C++ -*-
//
// File:        procregs.h
// Description: C definitions of selected PIC processor registers
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// PCM-C readable register definitions for
// general PICs.  This version is tailored
// only for the PIC16F84 and for the 12C672.
//

#byte   INDF    = 0x00
#byte   TMR0    = 0x01
#byte   PCL     = 0x02

struct {
  short int C;
  short int DC;
  short int Z;
  short int PD_L;
  short int TO_L;
  short int RP0;
  short int RP1;
  short int IRP;
} STATUS;
#byte   STATUS  = 0x03
#byte   FSR     = 0x04
#byte   PORTA   = 0x05
#byte   PORTB   = 0x06
#byte   PCLATH  = 0x0A
struct {
  short int RBIF;
  short int INTF;
  short int T0IF;
  short int RBIE;
  short int INTE;
  short int T0IE;
  short int PEIE;
  short int GIE;
} INTCON;
#byte   INTCON  = 0x0B

#endif
#list
```

## File: radio.h

```
#ifdef _RADIO_H
#nolist
#else
#define _RADIO_H
// -*- Mode: C++ -*-
//
// File:        radio.h
// Description: serial interface to TR1000 radio chip
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// warm reset code

#define RADIO_SETUP_TX()                \
        gRState = 0

#define RADIO_SETUP_RX()                \
        gRState = SRR_SYNC;             \
        PORT_B.RADIO_XMT = 0


// number of bit periods to try for a sync header before
// timing out.  each sync header is 6 bits long; try for
// 10 periods.
#define SYNC_TIMEOUT    60

// number of 6 bit sync chars to send when transmitting a
// header.
#define SYNC_HDR_LENGTH 6 // in 6-bit "hexlets"

// in receive mode, monitor RADIO_RCV line, read encoded serial
// data, decode, store in FIFO.
#inline
void service_radio_rcv();

// in transmit mode, fetch bytes from FIFO, encode and send as
// serial data on RADIO_XMT line
#inline
void service_radio_xmt();

#endif
#list
```

## File: radio.c

```
// -*- Mode: C++ -*-
//
// File:        radio.c
// Description: manage exchange of serial data with TR1000 radio
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// NOTE: send/receive MSB first
//
// The servicing of the host parallel port happens only in those bit
// periods where service_radio_rcv() doesn't have other time consuming
```

```
// operations, such as transfers to the fifo or encoding and decoding
// six bit values.  In short, timing is critical!
//
// 06 Nov 2000 r@media.mit.edu
//
// In our noisy environment, BART detects many false sync headers.
// Upon transmission, BART generates a leader char at the start of
// each packet.  Upon reception, if the first non-sync char isn't a
// leader char, BART reverts to searching for sync without troubling
// the host.  This should cut down on the number of spurious packets
// handled by the host.

#include "radio.h"
#include "procregs.h"
#include "utils.h"
#include "sync.h"
#include "fifo.h"

// ================================================================

// Variables can be safely shared between service_radio_rcv() and
// service_radio_xmt() since the system can't be receiving and
// transmitting at the same time.

int gRState;
int gRSerBuf;                   // six bit serial shift register
int gRBytBuf;                   // holding register for decoded byte

// Every packet has leader char immediately following the synch header
#define LEADER_CHAR 'A'         // got any better ideas?
short int gIsPacketStart;       // true when seeking leader char

#inline
void _get_radio_bit();

#inline
void _put_radio_bit();

// ================================================================
// service_radio_rcv() - in receive mode, monitor RADIO_RCV line,
// read encoded serial data, decode and store in FIFO.
//
// In the initial state, it will search for a sync pattern on the
// radio's receive line.  Once it establishes sync, it will read
// serial bits, decode their 6 bit form into a 4 bit nibble, assemble
// pairs of nibbles into bytes, and store the bytes in the FIFO.
//
// This routine essentially implements a software UART.  In all but
// the initial state, fewer than BIT_PERIOD cycles may elapse between
// calls to service_radio_rcv(), or data will be lost.
//

#inline
void service_radio_rcv() {
#asm
    SHORT_DISPATCH(gRState)
#define SRR_SYNC 0
      goto srr_sync                     // establish sync, sample b11 (msb)
#define SRR_B11_INITIAL 1
      goto srr_b11_initial              // b11 & service host port
#define SRR_B10_INITIAL 2
      goto srr_b10_initial              // b10 & service host port
#define SRR_B09 3
      goto srr_b09                      // b09 & service host port
      goto srr_b08                      // b08 & service host port
      goto srr_b07                      // b07 & service host port
      goto srr_b06                      // b06 & store high nibble
      goto srr_b05                      // b05 & service host port
      goto srr_b04                      // b04 & service host port
      goto srr_b03                      // b03 & service host port
      goto srr_b02                      // b02 & service host port
      goto srr_b01                      // b01 & service host port
      goto srr_b00                      // b00 & store low nibble
      goto srr_b11_store                // b11 & service host port
      goto srr_b10_store                // b10, accumulated byte to fifo
#endasm

 srr_sync:
    // Here to establish initial sync.  Note that unlike other
    // routines in service_radio_rcv() which all complete in
    // less than 44 tics, this one might take a relatively long
    // time to complete.  During this time, host transfers are
    // deferred.
    gIsPacketStart = 1;
    if (!find_sync(SYNC_TIMEOUT)) {
      // didn't find sync.  Try again at next call to service_radio_rcv()
      gRState = SRR_SYNC;
      return;
    }
    // found sync.  From here on, TMR0 rolling over (T0IF) marks the
    // midpoint of each received bit.  Fetch the first bit (b11, MSB
    // first) before returning.
    gRState = SRR_B11_INITIAL;  // next: srr_b10
    // --v-- fall through... --v--

 srr_b11_initial:
 srr_b10_initial:
 srr_b09:
 srr_b08:
 srr_b07:
    // b06 below
 srr_b05:
 srr_b04:
 srr_b03:
 srr_b02:
 srr_b01:
    // b00 below
 srr_b11_store:
    // sample a bit and service the parallel port
    _get_radio_bit();
    service_host_rcv();
    gRState++;
    return;

 srr_b06:
    // sample bit 6 of input stream.  Having accumulated the first
```

```
    // 6 bit packet, decode into 4 bit and store in accum byte.
    _get_radio_bit();
    codec_decode(gRSerBuf);
    // Following the call to codec_decode(), the decoded 4 bit nibble
    // is in the W register.  Do the equivalent to:
    // gRBytBuf = codec_decode(gRSerBuf);
    W_RECV(gRBytBuf);
    if (gRBytBuf == DECODE_SYNCH) {
      gRState = SRR_B11_INITIAL;        // strip sync bytes
    } else {
      gRState++;
    }
    return;

 srr_b00:
    // sample bit 00 (LSB), decode accumulated 6 bits to 4 bit,
    // merge with gRBytBuf.
    // ## Note that we don't check for sync or illegal bit patterns
    // ## here - if we get either, gRBytBuf is blithely clobbered.
    // ## On the other hand, the sending code will only generate
    // ## sync filler for the first nibble, so we don't expect to
    // ## get sync filler here.
    _get_radio_bit();
    swap(gRBytBuf);             // make room in lsnibble
    codec_decode(gRSerBuf);
    // After the call to codec_decode(), the decoded 4 bit nibble is
    // in the W register.  The following IORWF is equivalent to:
    // gRBytBuf |= codec_decode(gRSerBuf);
#asm
    iorwf gRBytBuf,f
#endasm
      if (gIsPacketStart) {
        // in srr_b06, sync "nibbles" were stripped out.  Arrive here
        // after finding the first non-sync character.  If it was a
        // bona-fide leader char, accept it and start reading the rest
        // of the packet.  If not, start all over again looking for
        // sync.
        if (gRBytBuf == LEADER_CHAR) {
          gRState = SRR_B11_INITIAL;    // strip sync bytes
          gIsPacketStart = 0;
          return;
        } else {
          // wasn't
          gRState = SRR_SYNC;
          return;
        }
      }
    gRState++;                  // sample b11 and store...
    return;

 srr_b10_store:
    // sample bit 10, store byte previously accumulated (gRBytBuf) in
    // the fifo.  We wait until bit 10 (rather than bit 11) so that the
    // service_host_rcv() is  called every other tic.
    _get_radio_bit();
    fifo_put(gRBytBuf);
    gRState = SRR_B09;
    return;
}

// ============================================================
// Wait for TMR0 to roll over, shift the radio serial port into
// the LSB of the serial buffer.  Update TMR0 before returning.
//
#inline
void _get_radio_bit() {
  AWAIT_TMR0();
  SHIFT_BIT_LEFT(gRSerBuf, PORT_A.RADIO_RCV);
  // output_high(PIN_B3);       // ## debug - wiggle b3 when sampling serial
  UPDATE_TMR0(BIT_PERIOD);
  // output_low(PIN_B3);        // ## debug
}

// ============================================================
// Serial Transmit (from FIFO to Radio)
//
// Initially, wait for a byte to appear in the fifo (await_start).
// Send a stream of sync chars (sync_b05-b00).  Thereafter, start
// sending bits from the fifo (send_b11-b00), MSB first.  If the fifo
// ever runs dry, send sync chars until more data is available.

// SYNC_CHAR defines the six bit sync header char, left justified in
// an 8 bit byte
#define SYNC_CHAR 0b11100000

#inline
void service_radio_xmt() {

#asm
  SHORT_DISPATCH(gRState)
#define AWAIT_START 0
    goto await_start           // s=00 loop until at least one byte in fifo
#define SYNC_B05 1
    goto sync_b05              // s=01 send sync bit 5, service host port
    goto sync_b04              // s=02 send sync bit 4
#define SYNC_B03 3
    goto sync_b03              // s=03 send sync bit 3, service host port
    goto sync_b02              // s=04 send sync bit 2, service host port
#define SYNC_B01 5
    goto sync_b01              // s=05 send sync bit 1, service host port
    goto sync_b00              // s=06 send sync bit 0
#define SEND_B11 7
    goto send_b11              // s=07 send bit 11 (msb), service host port
    goto send_b10              // s=08 send bit 10, service host port
    goto send_b09              // s=09 send bit 09, service host port
    goto send_b08              // s=10 send bit 08, service host port
    goto send_b07              // s=11 send bit 07, service host port
    goto send_b06              // s=12 send bit 06, encode low nibble
    goto send_b05              // s=13 send bit 05, service host port
    goto send_b04              // s=14 send bit 04, service host port
#define SEND_B03 15
    goto send_b03              // s=15 send bit 03, service host port
    goto send_b02              // s=16 send bit 02, fetch byte from fifo
    goto send_b01              // s=17 send bit 01, service host port
    goto send_b00              // s=18 send bit 00 (lsb), encode high nibble
#endasm
```

```
 await_start:
  // wait for first byte to appear in the fifo.
  gIsPacketStart = 1;
  service_host_xmt();           // service parallel port
  if (PORT_B.RCV_MODE) {
    // host is requesting receive mode.  Quit now.
    gXmtActive = 0;
    return;
  }
  if (!FIFO_IS_EMPTY()) {
    gRBytBuf = SYNC_HDR_LENGTH; // # of sync nibbles to send
    gRSerBuf = SYNC_CHAR;       // set up char to be sent
    SET_TMR0(BIT_PERIOD);
    // from here on, TMR0 marks onset of each bit period, honored by
    // _put_radio_bit()
    gRState++;                  // send sync 05
  }
  return;

 sync_b05:
 sync_b03:
 sync_b02:
 sync_b01:
 send_b11:
 send_b10:
 send_b09:
 send_b08:
 send_b07:
 send_b05:
 send_b04:
 send_b03:
 send_b01:
  _put_radio_bit();             // put next msbit of gRSerBuf
  service_host_xmt();           // service parallel port
  gRState++;
  return;

 sync_b04:
  // send sync bit 04 and use gRBytBuf to decide whether to keep
  // sending synch chars (sync_b03) or to switch to the mainstream
  // code (send_b03).  By switching to mainstream code, the final
  // 3 bits of the synch char will be sent, but the next byte
  // will be fetched from the FIFO, encoded and sent.
  _put_radio_bit();
#ifdef DONT_BUM_CYCLES
  if (--gRBytBuf) {
    gRState = SYNC_B03;
  } else {
    gRState = SEND_B03;
  }
#else
#asm
  movlw SEND_B03
  decfsz gRBytBuf, f
  movlw SYNC_B03
  movwf gRState
#endasm
#endif
  return;

 sync_b00:
  // put the last bit of gRSerBuf set state to send_b05 to send another
  // sync char
  _put_radio_bit();
  if (PORT_B.RCV_MODE && FIFO_IS_EMPTY()) {
    // host is requesting receive mode and the fifo has drained.
    // game over.
    gXmtActive = 0;
    return;
  }
  gRSerBuf = SYNC_CHAR;         // recharge gRSerBuf with sync char
  gRState = SYNC_B05;
  return;

 send_b06:
  // send bit 06 and decode the low nibble of gRBytBuf
  _put_radio_bit();
  swap(gRBytBuf);               // encode low nibble of gRBytBuf
  codec_encode(gRBytBuf);       //   into 6 bit value
  W_RECV(gRSerBuf);
  gRState++;                    // next state = send_b05
  return;

 send_b02:
  // send bit 02 and fetch the next byte from the fifo.  If the fifo
  // is empty, send a synch char after sending bit 0
  _put_radio_bit();
  if (FIFO_IS_EMPTY()) {
    gRBytBuf = 1;
    gRState = SYNC_B01;
    return;
  }
  if (gIsPacketStart) {
    // the very first char in the packet is the leader char
    gRBytBuf = LEADER_CHAR;
    gIsPacketStart = 0;
  } else {
    fifo_get();
    W_RECV(gRBytBuf);   // fifo_get() returns val in W
  }
  gRState++;            // next state = send_b01
  return;

 send_b00:
  // send bit 00 and encode the high nibble of the next byte
  _put_radio_bit();
  swap(gRBytBuf);               // encode high nibble of gRBytBuf
  codec_encode(gRBytBuf);       //   into 6 bit value
  W_RECV(gRSerBuf);
  gRState = SEND_B11;
  return;

}

// ================================================================
```

```
// Wait for TMR0 to roll over, shift out the MSB of the serial buffer
// (gRSerBuf) and write it to the radio port.  Update TMR0 before
// returning.  Note that gRSerBuf is not only shifted, but in the #asm
// code, may accumulate garbage in its right half.
//
#inline
void _put_radio_bit() {
  AWAIT_TMR0();
#ifdef DONT_BUM_CYCLES
  output_bit(shift_left(&gRSerBuf, 1, 0));
#else
#asm
  rlf gRSerBuf,f
  btfss STATUS.C
  bcf PORT_B.RADIO_XMT
  btfsc STATUS.C
  bsf PORT_B.RADIO_XMT
#endasm
#endif
  UPDATE_TMR0(BIT_PERIOD);
}
```

# File: sync.h

```
#ifdef _SYNC_H
#nolist
#else
#define _SYNC_H
// -*- Mode: C++ -*-
//
// File:        sync.h
// Description: establish byte and bit level synch with radio
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

void send_sync(int n);
void puthexlet(int hexlet);
void putnibble(int nibble);

// Try for n bit periods to find a sync header.  If found, try to
// establish bit level sync.  Returns 1 with TMR0 primed to roll over
// in the middle of the first bit period, returns 0 otherwise.
//
int find_sync(int n);

#endif
#list
```

# File: sync.c

```
// -*- Mode: C++ -*-
//
// File:        sync.c
// Description: generate and detect radio packet sync
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

// Generating and detecting sync.
//
// Each transmitted packet starts with a sync header, which is
// a string of sync header characters.  Each sync header char
// is 111000, that is, three bit periods on and three off.
//
// The advantage of this particular pattern is that it features
// a 50% DC balance and a single low-to-high transition in the
// middle of the pattern.

// ================================================================
// receiving sync

// Try to establish sync.  If no SYNC_HEADER pattern is discovered
// within N bit periods, the routine returns 0 to indicate failure.
// If a sync header is found, the routine has established byte level
// synch, but not fine-grained bit level sync.  It then spends a
// deterministic amount of time (to be documented :) adjusting the
// bit level sync.
//
// Upon success, find_sync() will return 1 and TMR0 will be set up
// to roll over in the middle of the first bit period of the next
// "hexlet" (a six bit nibble).

// Theory:
// The SYNC_HEADER is a rectangular wave, three bit periods
// high followed by three bit periods low, or 0x0f when read
// from LSB first.  One bit period is 44 clock tics long.
//
// find_sync() samples once every 44 tics, shifting received bits into
// a register.  When the bit pattern 00011100 is detected, then the
// header is assumed to fall as follows:
//
//     >|   |< 44 tics
//              |---|---|---|           |
//  |---|---|---|           |---|---|---|
//    |   |   |   |   |   |   |   |   |
//    0   0   0   1   1   1   0   0
//    B7  B6  B5  B4  B3  B2  B1  B0
//  (msb)

// This is the 8 bit pattern that will be accumulated in tmp
// when timing is between BIT_PERIOD and 2*BIT_PERIOD tics
// from the onset of the next SYNC_HEADER.

#define PRE_SYNC 0b00011100

int find_sync(int n) {
  int tmp, i;

  SET_TMR0(BIT_PERIOD);

  // phase 1: Establish byte level synchronization by shifting in bits
  // until SYNCH_HEADER is detected (or until we exceed our alloted
  // number of bit cell times).

  tmp = 0;
  // output_high(PIN_B3);               // ## debugging
  do {
    restart_wdt();
    AWAIT_TMR0();
    SHIFT_BIT_LEFT(tmp, PORT_A.RADIO_RCV);
    UPDATE_TMR0(BIT_PERIOD);
    if (!--n) return 0;          // timed out
  } while (tmp != PRE_SYNC);
  // output_low(PIN_B3);

  // At this point, the picture looks like this.  B0 may have been
  // sampled as early as 2*BIT_PERIOD before the onset of the next
  // SYNC_HEADER frame:
  //                  |----|----|----|              |
  //  |----|----|----|              |----|----|----|
  //    |    |    |    |    |    |    |    |        ^
  //    0    0    0    1    1    1    0    0      onset
  //   B7   B6   B5   B4   B3   B2   B1   B0
  //
```

```
  // or as late as 1*BIT_PERIOD tics before the onset:
  //                 |----|----|----|              |
  //  |----|----|----|              |----|----|----|
  //      |    |    |    |    |    |    |    |    ^
  //      0    0    0    1    1    1    0    0   onset
  //      B7   B6   B5   B4   B3   B2   B1   B0

  // At this point, DLY cycles have elapsed since sampling B0.  So
  // if B0 was sampled early, the onset of the next synch frame will
  // be in 2 * BIT_PERIOD - DLY sample hence.  If B0 was sampled late,
  // the onset of the next frame will be BIT_PERIOD - DLY cycles from
  // now.
  //
  // All that's left to do is to look for the low-to-high transition
  // that marks the next sync frame, which is expected no sooner than
  // BIT_PERIOD-DLY and no later than 2*BIT_PERIOD-DLY cycles from now.
  //
  // At 44 cycles per bit period, we expect the signal to remain low a
  // minimum of 44-DLY cycles and a maximum of 88-DLY.
  //
  // DLY measured to be 25, so min = 19, max = 63.
  // The following code is written in assembly to guarantee the cycle
  // counts.

#asm
                              // cycle
    btfsc PORT_A.RADIO_RCV    // 00
    goto early                // 01
    btfsc PORT_A.RADIO_RCV    // 02
    goto early                // 03
    btfsc PORT_A.RADIO_RCV    // 04
    goto early                // 05
    btfsc PORT_A.RADIO_RCV    // 06
    goto early                // 07
    btfsc PORT_A.RADIO_RCV    // 08
    goto early                // 09
    btfsc PORT_A.RADIO_RCV    // 10
    goto early                // 11
    btfsc PORT_A.RADIO_RCV    // 12
    goto early                // 13
    btfsc PORT_A.RADIO_RCV    // 14
    goto early                // 15
    btfsc PORT_A.RADIO_RCV    // 16
    goto early                // 17
    btfsc PORT_A.RADIO_RCV    // 18
    goto early                // 19
    btfsc PORT_A.RADIO_RCV    // 20
    goto found                // 21
    btfsc PORT_A.RADIO_RCV    // 22
    goto found                // 23
    btfsc PORT_A.RADIO_RCV    // 24
    goto found                // 25
    btfsc PORT_A.RADIO_RCV    // 26
    goto found                // 27
    btfsc PORT_A.RADIO_RCV    // 28
    goto found                // 29
    btfsc PORT_A.RADIO_RCV    // 30
    goto found                // 31
    btfsc PORT_A.RADIO_RCV    // 32
    goto found                // 33
    btfsc PORT_A.RADIO_RCV    // 34
    goto found                // 35
    btfsc PORT_A.RADIO_RCV    // 36
    goto found                // 37
    btfsc PORT_A.RADIO_RCV    // 38
    goto found                // 39
    btfsc PORT_A.RADIO_RCV    // 40
    goto found                // 41
    btfsc PORT_A.RADIO_RCV    // 42
    goto found                // 43
    btfsc PORT_A.RADIO_RCV    // 44
    goto found                // 45
    btfsc PORT_A.RADIO_RCV    // 46
    goto found                // 47
    btfsc PORT_A.RADIO_RCV    // 48
    goto found                // 49
    btfsc PORT_A.RADIO_RCV    // 50
    goto found                // 51
    btfsc PORT_A.RADIO_RCV    // 52
    goto found                // 53
    btfsc PORT_A.RADIO_RCV    // 54
    goto found                // 55
    btfsc PORT_A.RADIO_RCV    // 56
    goto found                // 57
    btfsc PORT_A.RADIO_RCV    // 58
    goto found                // 59
    btfsc PORT_A.RADIO_RCV    // 60
    goto found                // 61
    btfsc PORT_A.RADIO_RCV    // 62
    goto found                // 63
    goto late                 // expected a transition bit by now
#endasm

 early:
 late:
    return 0;

 found:
    // Arrive here within 5 (min) to 7 (max) cycles of RADIO_RCV going
    // true.  Assume it was 6, set TMR0 to roll over (BIT_PERIOD/2)-6
    // cycles from now
    //
    // ## NB: If this doesn't leave enough time in the caller's code to
    // ## prepare for the bit, this routine could be modified to detect
    // ## the middle of the SYNC_HEADER rather than the end (the falling
    // ## edge rather than the rising edge), and set TMR0 to fire in
    // ## (3.5 * BIT_PERIOD - 6) rather than (0.5 * BIT_PERIOD - 6)
    //
    // output_high(PIN_B3);     // ###
    SET_TMR0((BIT_PERIOD/2)-6);
    // output_low(PIN_B3);      // ###
    return 1;
}
```

# File: utils.h

```
#ifdef _UTILS_H
#nolist
#else
#define _UTILS_H
// -*- Mode: C++ -*-
//
// File:        utils.h
// Description: generally useful code hacks for the PIC
//
// Copyright 2001 by the Massachusetts Institute of Technology.  All
// rights reserved.
//
// This MIT Media Laboratory project was sponsored by the Defense
// Advanced Research Projects Agency, Grant No. MOA972-99-1-0012.  The
// content of the information does not necessarily reflect the
// position or the policy of the Government, and no official
// endorsement should be inferred.
//
// For general public use.
//
// This distribution is approved by Walter Bender, Director of the
// Media Laboratory, MIT.  Permission to use, copy, or modify this
// software and its documentation for educational and research
// purposes only and without fee is hereby granted, provided that this
// copyright notice and the original authors' names appear on all
// copies and supporting documentation.  If individual files are
// separated from this distribution directory structure, this
// copyright notice must be included.  For any other uses of this
// software, in original or modified form, including but not limited
// to distribution in whole or in part, specific prior permission must
// be obtained from MIT.  These programs shall not be used, rewritten,
// or adapted as the basis of a commercial software or hardware
// product without first obtaining appropriate licenses from MIT.  MIT
// makes no representations about the suitability of this software for
// any purpose.  It is provided "as is" without express or implied
// warranty.

#include "procregs.h"

// ================================================================
// DISPATCH

// Efficient dispatch table.  Takes 11 cycles (including goto...).
// Offset is clobbered.  Must be inside #asm context
//
#define DISPATCH(offset)        \
  movplw jmptbl                 /* offset += low byte of jmptbl addr    */ \
  addwf offset,f                /*   carry set if crossing page bounds  */ \
  movphw jmptbl                 /* w = high byte of jmptbl addr         */ \
  btfsc STATUS.C                /* if (carry was set in addwf)          */ \
    addlw 1                     /*      w += 1                          */ \
  movwf PCLATH                  /* pclath = w                           */ \
  movf  offset, w               /* w = offset                           */ \
  movwf PCL                     /* pcl = w                              */ \
jmptbl:

// Shorter DISPATCH(), but valid iff table falls within current page
// (doesn't increment PCLATH).  Takes 7 cycles (including goto ...)
// must be inside #asm context
//
#define SHORT_DISPATCH(offset)  \
  movphw jmptbl                                                         \
  movwf PCLATH                                                          \
  movf offset, w                                                        \
  addwf PCL, f                  /* pcl += offset        */              \
jmptbl:

// ================================================================
// Working with Timer 0 (aka RTCC)
//
// Primarily intended for critical timing loops, these macros
// assume that the TMR0 prescaler is set to "DIV_1" and counts
// once every instruction cycle.

// Set TMR0 to roll over after a given number of tics.  The
// +2 term accounts for a two cycle inhbit when TMR0 is set.
//
#define SET_TMR0(tics) TMR0 = (256+2-(tics));  INTCON.T0IF=0

// Set TMR0 to roll over tics counts AFTER the previous roll over.
// After an initial call to SET_TMR0(), you can use UPDATE_TMR0()
// to prevent accumulating timing errors.
#define UPDATE_TMR0(tics) TMR0 += (256+2-(tics)); INTCON.T0IF=0

// Busy wait for the Timer 0 Interrupt Flag (T0IF).  Assumes TMR0 has
// been set and that T0IF has been cleared by means of a previous call
// to SET_TMR0() or UPDATE_TMR0().
//
#define AWAIT_TMR0() while (!INTCON.T0IF)
//
// Bug catching version.  It takes a few extra precious cycles, but
// jumps to damn() with the caller's PC if INTCON.T0IF was set when
// AWAIT_TMR0() was first called.
//
// #define AWAIT_TMR0() if (INTCON.T0IF) damn(); while (!INTCON.T0IF)
// int gLoss;
// void damn() { gLoss++; }

// make TMR0 roll over delta tics sooner than scheduled
#define ADVANCE_TMR0(delta) TMR0 += ((delta)+2)

// make TMR0 roll over delta tics later than scheduled
#define RETARD_TMR0(delta) TMR0 -= ((delta)-2)


// ================================================================
// Favorite time savers

// shift srcBit into the LSB of dstByte
// equivalent to shift_left(&dstByte, 1, srcBit)
#define SHIFT_BIT_LEFT(dstByte, srcBit)         \
    STATUS.C = srcBit;                          \
```

```
#asm                                        \
    RLF dstByte, F                          \
#endasm                                     \
    {}

// shift srcbit into the MSB of dstByte
// equivalent to shift_right(&dstByte, 1, srcBit)
#define SHIFT_BIT_RIGHT(dstByte, srcBit)    \
    STATUS.C = srcBit;                      \
#asm                                        \
    RRF dstByte, F                          \
#endasm                                     \
    {}


// ============================================================
// Using W for passing arguments

// direct access to W register, useful for passing single
// args to functions.  eg:
//      void wputc() {
//        int ch;
//        W_RECV(ch)
//        ...
//      }
//
//      ...
//        W_CALL_LIT(wputc, 'a');

#define W_PASS(arg)             \
#asm                            \
  movf  arg, w                  \
#endasm

#define W_PASSL(arg)            \
#asm                            \
  movlw arg                     \
#endasm

#define W_RECV(arg)             \
#asm                            \
  movwf arg                     \
#endasm

#define W_CALL(func, arg)       \
#asm                            \
  movf  arg,w                   \
#endasm                         \
  func()

#define W_CALL_LIT(func, arg)   \
#asm                            \
  movlw arg                     \
#endasm                         \
  func()

#endif
#list
```