The Prototype Embedded Network (PEN)

Alan Jones[†] and Andrew Hopper[†‡]

[†] AT&T Laboratories Cambridge Limited
24A Trumpington Street, Cambridge CB2 1QA, England

[‡] Laboratory for Communications Engineering
University of Cambridge Department of Engineering
Trumpington Street, Cambridge CB2 1QA, England

Abstract

This paper describes a prototype low-power wireless link designed for embedding into everyday objects. After a brief description of the hardware platform, we discuss protocols for discovery and long-term rendezvous, contrasting our approach with master based approaches. The protocols are presented as examples drawn from a larger family of interoperable protocols which may be important to cover the broad range of applications for this sort of network. We then describe some of the applications built using PEN, and add some observations on spectrum costs, and future designs.

Keywords

Wireless, Ad-hoc, Embedded, Network, Protocol

Motivation

Embedded networks may be built for many purposes, and it is not clear where they will first reach widespread deployment. By way of introduction we will set them in the context of providing an electronic means of sensing what is around, so that devices can configure themselves and operate autonomously with less input from their owners.

Computers are being embedded in a growing number of everyday objects. Often, this is revealed by the introduction of new features that can be added at little extra cost, such as clocks, alarms and timers, memories to recall different settings, and security codes. The problem is that all of these new features need setting, and resetting. There is a danger that as computers are incorporated into more systems to make them more efficient and easy to use, we may paradoxically find ourselves spending more time in configuring and maintaining them. One possible solution to this dilemma is to make the devices more capable of configuring themselves. This could be achieved by allowing them more access to information about their environment, such as where they are, indoors or out, who is around and what they are doing. Of course, some systems already incorporate some environmental sensing – room thermostats, smoke alarms, PIR detectors for lights and alarms, but these are usually hard wired into their systems with little opportunity for sharing information. We would like to see them supplied as components in a more modular architecture for sharing and acting on information about our habits.

We are not advocating complex recognition tasks such as deducing who is present or what they are doing by the interpretation of images, sounds etc. Rather we favour sensors that have a direct interpretation, so if we need to know who is operating a device or entering a room, then we will ask them to tell us by pressing a button or carrying a tag. This minimal requirement not only removes the need for complex recognition tasks that often require human perceptual talents, but can also put the users more in control as the system is seen to respond only when they identify themselves to it, and does not act at unexpected times.

An example of how more open interfaces could simplify equipment use would be a status input for the home. This could take the form of a switch with four options – at home, out for a short while, gone to bed, and away. Devices that might benefit from this simple piece of information include lights, heating, alarms, entertainment units, telephones and answering machines. They would know to be quiet and save energy when we were in bed, to keep things ready for our return if we are out for a short time, and to shut down safely and securely if we are away. Of course, some lights might still come on while the house is unoccupied to give the impression of habitation, and smoke and burglar alarms would call for outside help more quickly when the owner is away. Smoke alarms could also call for lights to be put on and external bells to be rung. However, more advantages can be realised when devices communicate. Smoke alarms, lighting and heating controls could all benefit from the separation of the sensors (detectors, switches, knobs) and the actuators (lighting relays, boiler valves, alarm bells). The examples are too numerous to mention, but instances might be that smoke detected in any part of the house could set off all of the alarms, a defective unit (commonly caused by batteries being removed for some other appliance and not replaced) could be detected by its absence, and others units could signal the problem.

One obstacle to the deployment of such systems is the amount of wiring involved. In some cases, such as burglar alarms, the number and location of sensors and bells is small enough that the extra wiring can be fitted in a few hours by a skilled installer, but the cost of fitting may still exceed the cost of the hardware itself. Also, as more sensors and actuators are added, the tolerance of the householders to exposed wiring will be tested, and burial into walls, ceilings and floors will be required. For this reason, we believe that this will be a good application area for wireless embedded networks.

Hardware

The design of our prototype embedded network (PEN) was guided by the minimisation of three main parameters: size, power consumption, and cost. This should allow us to embed it into many devices with a minimal impact on their appearance, longevity and saleability. An early version of the system was described in [Bennett et al 97].

Size and cost are minimised by keeping the design as simple as possible, so that it can be readily integrated, and so that the minimal implementation does not require a processor to run the radio or respond to simple requests. A test for any proposed protocol change is that we can still build a node with a simple state machine for recognising and responding to packets addressed to it. This does not rule out more complex nodes as will be described later, but they must be able to work with our most trivial sensors and beacons. The radio is kept simple by having all nodes transmit and receive on the same frequency, assuming a low channel utilisation, simple modulation, and restricted range of around 5 metres.

The nodes described in this paper are based on a 418MHz FM transceiver, a Toshiba microcontroller, and a Xilinx programmable logic device. The standard node is equipped with SRAM and Flash memory, a piezo sounder, LEDs, buttons and serial communications ports (see figure 1). As they are prototypes, designed to facilitate a wide range of experiments, they are larger and more complex than any one application would require.

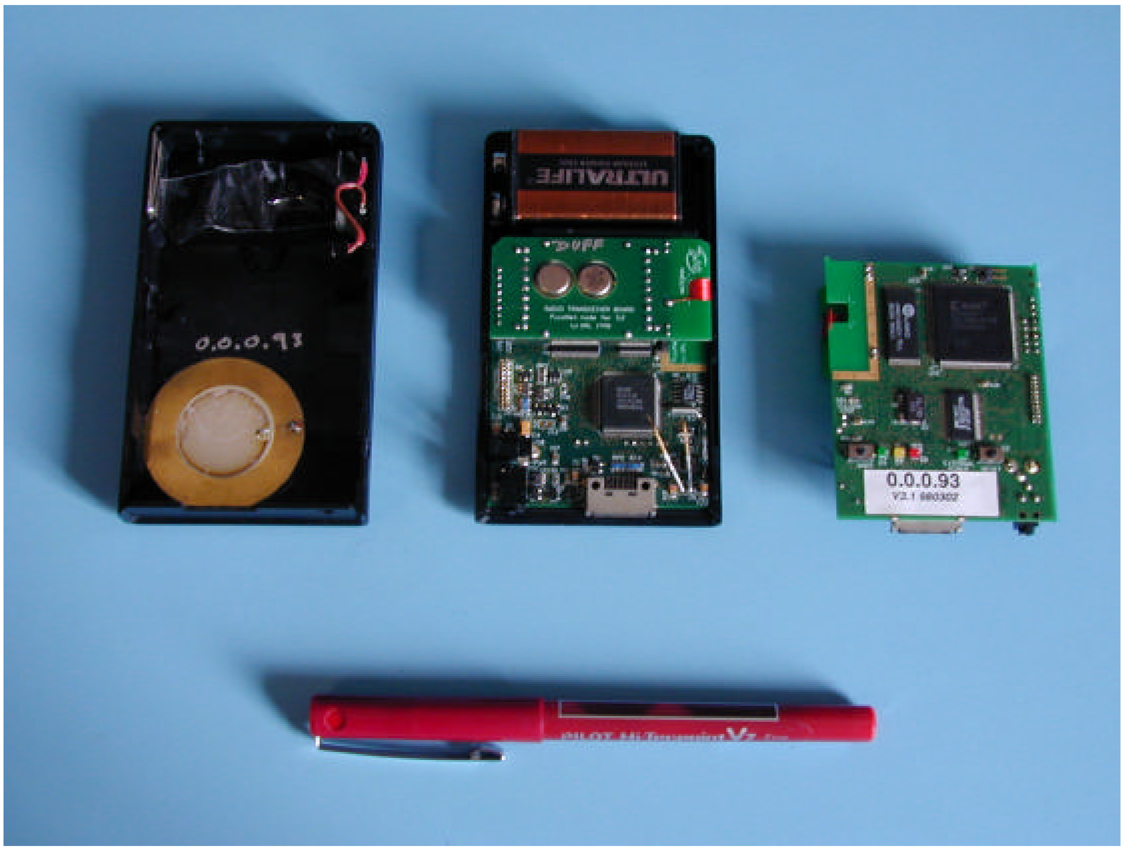

Figure 1. Both sides of a PEN node, and a pen.

Power consumption is minimised by ensuring that nodes can be completely off for most of the time. We believe that within a few years, we will be able to run a radio transmitter or receiver with 1mA at 1V, and a simple processor or state machine for 2mA at 1.5V. If we are only active for 5ms every 10s, then we are using 4mW at a duty cycle of 1 in 2000, and could achieve an average power consumption of 2 uW. If we allow for modem and real-time clock, we might use 3uW on average. A CR2032 lithium coin cell weighing 3.3g has a capacity of 180mAh at 3V, and would power a node at 3uW for 180,000 hours, or 20 years. Allowing a further 3uW for leakage and other contingencies, we could achieve an acceptable lifetime of ten years before battery replacement, by which time the design should be obsolete (cf. pocket calculators and digital watches). To achieve this, we will have to limit the leakage currents in deep sleep by selection of devices that support very low current sleep, or by isolating the power supply lines. We will also need to minimise the capacitances that must be charged when coming out of sleep, preferably by choosing design techniques that do not require much decoupling.

Our present hardware built with off-the-shelf components has a peak power consumption of 600mW. A typical application that listens for a free channel, transmits a beacon packet, and listens for replies every 20s, has to be active for 600ms every 20s, a duty cycle of 1 in 30. This is an average power of 20mW, and a lifetime of 450 hours (20 days) on a 9Whr battery, which agrees with our observations. This poor lifetime is most easily improved by reducing the radio turn-on and preamble times so that the beacon only requires us to be active for 60ms, which should improve the lifetime to 200 days. Our sleep current is 16uA, which should still consume less than 10% of the battery in 200 days.

Sensors will increase the power requirements for some nodes, leading to correspondingly shorter lives, or larger batteries. Powering from solar energy or mechanical sources may be an option for some applications, though as with digital watches, a battery is probably going to be the most common power source.

Discovery

The nodes are assumed to be powered down, completely unresponsive, except for a wakeup timer, for most of the time. For two nodes to discover each other's presence, they must both be awake at the same time. The simplest way in which this can be achieved is to have one of them enter a high power mode in which it stays awake for a time interval greater than the time between wakeups. There are then two options, have one node (the enquiring node) transmit continuously and the other (the sleeping node) listen occasionally, or vice versa.

If the enquiring node transmits continuously, it will be occupying the radio channel, and we would not achieve the low channel utilisation that we desire. If it receives continuously, then the sleeping nodes will have to transmit a short packet whenever they wake up. We have adopted this latter scheme as it scales well to large collections of nodes, and allows us to discover many sleepy nodes in one interval. If nodes expect to enter into two-way communications, then the sleepy nodes will have to listen for a short while after their transmissions so that other nodes can respond to them and negotiate a longer-term connection. The discovery protocol is completed by realising that we must listen before the transmission to make sure that the channel is free, so the full cycle is

Sleep (typically 10s)
Listen for free channel (typically 1ms)
Transmit signal for other nodes (typically 2ms)
Listen for replies (typically 2ms, optional)

We call this {listen, signal, listen} cycle a beacon, and the period with which it repeats is the beacon interval.

The node that is listening continuously to pick up the beacons may be able to do this because it has higher power resources, or it may be triggered temporarily by some other condition, for example a button press, a sensor input, a time of day or even a change in the other nodes with which it is maintaining contact. Some of the issues surrounding the choice of which nodes should listen and when are covered in [Todd et al 00].

Now, what should the transmitted signal contain?

1. Minimal. If we put no information in our signal other than its presence, then another node hearing it will have to reply to us on every occasion to find out if we are a node that it is interested in. This will waste power for both nodes, but would be the lowest power strategy while there are no other nodes around.

2. Node ID. If we put just a globally or locally unique ID in our signal, then other nodes will have to reply and ask for more information when they first encounter us, but they can then cache this information, and need not enquire again when they see the ID. This is a good strategy when the nodes around us do not change very often.

3. Service ID. If we put information in our signal about what services we offer or request, then other nodes can decide if they want to communicate with us without any further interaction. However, we pay the penalty of a longer signalling packet on every occasion. This strategy is best suited to environments where there is a high degree of mobility of the nodes, or where we wish to pay the power penalty in our node to conserve power in the other nodes around us. In high mobility situations, it allows frequent new sightings to be ruled out quickly as uninteresting, and if we find that we are interested, it has already cut down on the number of transactions.

4. Data Finally, we may put the service data itself in our signal, a temperature or location for example. This would be good in situations where the service offering is fixed, and where we do not mind using the extra bandwidth and power that it consumes. The simplest sensors may take this form, as they need only listen for the channel to become free, then transmit their sensor data, then sleep. They do not need to decode any incoming packets.

We have generally implemented option 3, but note that all four options can coexist. Further, they can interwork if nodes run a simple information seeking algorithm as follows:

- Listen to the channel for any PEN packets
- If any transmissions are received at any time, process them for ID, service offerings and requests, and service data.
- If we receive a transmission without an ID, ask the node for its ID.
- If we receive a node ID, and we don't have a note of what its services offerings or requests are, ask it for its offerings and requests.
- If we can supply what it requests, or want its offering and don't already have it, set up an interaction to satisfy that service.

The service interaction may be a simple data transfer, an agreement to send regular updates, or the activation of higher level protocols such as a low power transport layer.

This can be summed up by goal seeking for ID, service descriptions and service data until we find that we are not interested, or we have our information. If we cache what information we receive, then we do not have to ask again, but if the cache has flushed, we ask again.

The node that is beaconing can adapt to the recent interactions. If no other nodes are responding to its service descriptions, then it can fall back to ID only, and nearby nodes will hopefully have cached its service descriptions and will not respond. Later, it can fall back to an empty signal only, in which case nearby nodes will respond with a request for its ID. If no other nodes are around, then this node can continue with the empty signal, thus conserving its energy as much as possible. As soon as a node responds, it should resume putting the ID in the signal, so that the nearby nodes do not need to enquire every time. After some number (n) of beacon cycles, say 10, it can lapse back to signal only which will probe to see if there are still other nodes around, with the small expense of the extra transaction every n beacons.

Media Access Layer

The MAC layer will have to allow for a variable sized packet, with optional addressing fields to hold the node IDs of the sender and the destination. The channel is inherently a broadcast medium, but packets can be filtered at the receiver by source and destination node IDs, service offerings and requests, and service data.

A fundamental issue to be addressed is that of multiple replies to a packet. In each beacon packet, we include an element that defines a time window to be used for replies, which can be set by the application, or adapted to current conditions. Nodes wishing to reply will choose a time in this window at random and will have to wait for the next beacon if they collide. In the case of a packet that was addressed to a particular node ID, an acknowledgement may be requested by a bit in the header, and in this case there is an extra window immediately following the packet, which only the destination node should transmit in. The MAC layer is described in more detail in [Girling et al 00].

Long Term Rendezvous

Once nodes have discovered each other's presence, they may wish to stay in contact to transfer information. This could allow them to capture new sensor data, or to confirm that they are still in the same place from a location beacon, or to check regularly whether their services are required. To conserve power, it is desirable that both nodes sleep for most of the time, and wake occasionally for the shortest possible time to give the opportunity for further communication. If the discovery beacons from the previous section are known to occur at fixed intervals, then other nodes can sleep until just before they are due, listening for only a short time to confirm the continued presence of the beaconing node. If such nodes want a lower latency between contacts, then they can request a shorter beaconing interval, but this should last for only a limited number of cycles, and be renewed regularly. That way, we avoid the drain on resources when nodes have moved out of contact without rescinding their requests. This also applies to requests for extra data or regular sensor readings, they should always be accompanied by a count which will time them out unless renewed.

In some applications, a group of nodes may wish to stay in regular contact for an extended period. One option for this is to bring them all into some sort of synchronisation, either within a master framing sequence, or by bringing together their individual beaconing times. This sort of scheme can give rise to problems when nodes are entering and leaving regularly, because of the difficulty of reaching agreement about schedules in an environment where the communication links between some nodes are unreliable. A more interesting problem is how to bring together the schedules of two groups of nodes when some of their members begin to interact, without trying to pull the whole connected world into synchronisation. So long as they are identified as two distinct groups, then they can have separate schedules, but this may introduce artificial "membership" decisions into groups of nodes that do not require it, and the protocols to negotiate membership will burn yet more energy and bandwidth.

Our approach to long term rendezvous of groups of nodes is to keep all nodes equal. We allow the node that is beaconing to decide when those beacons should be transmitted. Other nodes can request shorter or longer intervals between beacons, but not the precise times when those beacons should appear. Thus a node that is listening to several other nodes will find itself having to wake up on a different schedule for each of those nodes. This may consume more power than a system where all the nodes transmit back-to-back in known slots, but this should not be significant if the radio, modem and processor are designed for fast turn-on.

Now that beaconing nodes are transmitting to a schedule that is known by other nodes, we are left with a problem of contention. If the beaconing node finds that the channel is not free, it should wait until the channel is free and then apply its usual backoff strategy to contend for access. Nodes that have woken up to hear the delayed beacon have several options. If they hear a contending packet, they can sleep until it ends, then listen for the beacon. If they do not hear a contending packet, the beaconing node may still be held off by a packet that only it can hear, so they should continue to listen for a time interval of one or two packet lengths (PEN imposes a maximum length on all packets). Alternatively, if they are aggressively power saving, or do not care about a few missed beacons, then they can immediately give up on this beacon and listen again for the next one. Only after several failures will they fall back to the discovery protocol to reacquire the beacons if the nodes are still around. For this to work well, the beaconing node must try to keep to its original schedule. If the beaconing node is blocked, it tries to return to its schedule for the next beacon. It will do this for a few tries, but then reschedule its beacon times from the next successful transmission. It can optionally alert other nodes by a control field in its header that it is going to permanently slip its beacon times, which may avoid them having to run the full discovery protocol.

Nodes that are conserving power do not necessarily have to listen to every beacon on schedule. They may choose to listen to every fourth beacon to maintain good timing synchronisation, but still have the option of communicating more quickly if triggered by some external event. Similarly, a beaconing node may choose to only transmit in every fourth interval when there is nothing timely to report, but would use every interval when there is activity to be reported. This scheme will save bandwidth and transmit power, but also the nodes that are listening at every interval can assume that there is nothing to come after a short period, and power down their receivers and processors sooner than if they were receiving a packet.

Comparison with Synchronised Protocols

One of the distinguishing features of the PEN protocols is that they do not attempt to synchronise the transmissions of a cluster of nodes by the use of a master station or global clock. We will discuss some of the advantages and disadvantages of these approaches.

Contention

In PEN, every initial transmission must be preceded by listening for the channel to be free, and each carries the possibility of collision if the transmissions are begun close enough together. The detection of the free channel can be made quite energy efficient by careful design of the radio link for fast detection and turn around, but if channel utilisation were high, then much energy could be wasted while waiting for other packets to finish. We deliberately avoid high utilisation of the channel in all of our protocols, and back off enthusiastically when contention is detected. After the contending packet and its acknowledgement have finished, there will be some slots for efficient contention to gain the next use of the channel. We hope that the energy overheads due to contention can be kept below 10%. The PEN protocols will require a channel capacity of about five times their maximum utilisation for efficient collision avoidance.

A master synchronised protocol avoids these problems by scheduling the transmissions of slave nodes so that they cannot collide [Bluetooth]. However, this approach requires extra communications with the master, well-synchronised clocks, and a low rate of change in connectivity. It may be that in real situations with lost packets, variable link quality, and mobility, the master/slave protocols use similar amounts of energy in their control packets, framing sequences and protocol overheads as PEN does in collision avoidance. In addition, if multiple masters and their slaves are to coexist, there must be some mechanism for synchronisation between masters, which usually involves reducing the bandwidth available to each master with a TDMA, FDMA or CDMA colouring scheme, also requiring a much higher total channel capacity than each master is allowed to use.

In both protocol approaches, nodes can sleep for most of the time, waking just in time to make their next scheduled transmission or reception. In a master/slave system, the master decides on the detailed scheduling of all of its slave nodes, whereas in PEN, it is the initiator of the radio transmission that decides its own schedule, and any listeners must fall in with that. In PEN, if a node is maintaining contact with several others, then it will transmit on its own schedule, and listen for the other nodes on theirs. If the schedules of two nodes begin to overlap, then in most cases, the later node will hear the transmission of the earlier, and wait until that has finished before sending its own. After a few such delays, the later node will slip its schedule permanently to a later time, and the overlap will have gone. In the rarer case of a collision where both nodes hear a free channel and begin to transmit together, then this may be detected by the lack of an expected acknowledgement, causing a retransmission after a random backoff, or in the case of unacknowledged transmissions, the situation will only be resolved when the difference in the clocks causes the nodes' schedules to differ sufficiently for one to hear the beginning of the others transmission. If the critical collision time is 100us, then the clocks might be expected to drift this far in 100s (assuming clocks that drift around 1 part in 1 million), so the collisions would not resolve for up to two minutes. If this were found to be a problem, then it may be worthwhile introducing a little random delay to the transmit times to reduce the probability of repeated collisions. In master/slave protocols, there will also be the possibility of collision when slaves first introduce themselves to masters, but this is a far less frequent operation, and the lack of an acknowledgement from the master soon informs the slaves to try again with some random variation.

The advantage that PEN gains from having no master protocol synchronising a cluster of nodes is that any pairs of the nodes can be separated from the others, physically or just because of the peculiarities of the radio channel, and they will still maintain any communications that were set up between them. In master/slave protocols, they would be reliant on signals from the master to schedule their transmissions, or even to pass their traffic for them.

Introductions

Another important function of the master may be to centralise information about what other nodes are around. In PEN, we perform this function with a proximity store. This is a service where one node keeps a list of other nodes whose beacons it has recently seen, and can make this list available to other nodes. With the node IDs, it also stores their beacon intervals, and the last time a beacon was seen. It can optionally store the service offerings and requests if they have been seen. If it is a high powered (listener) node, it can respond to the beacons of low powered node so that they can enquire about the nodes that they are interested in. If a low powered node wishes to communicate with a node described by a proximity store, then it can compute its next scheduled beacon time, sleep until then, and responding to the beacon directly to establish contact. In this way, we preserve the option of the two nodes to move away from the proximity store once contact has been made, we preserve the property that connectivity implies proximity, and we have not imposed any liveness or correctness constraints on the proximity store as its contents are used solely as hints to facilitate a more direct rendezvous. By separating out the functions of the proximity store into an optional service within the network, we can avoid imposing any overheads on applications that do not require them.

In a master/slave system, communication between slave nodes that have different masters is likely to involve some complexity, either to bring together all slaves that wish to intercommunicate under one master, or to have the masters relay traffic between themselves. In PEN, this is not a concern, as each node talks directly to those others that it wishes to communicate with.

Routing in PEN is an optional service, not part of the core protocols. [Weatherall et al 99]. We leave it up to the application programmer to decide if services should be loaded to perform routing of third party traffic between nodes, or via a wired network. We do not pay any code or energy penalties when these functions are not required, and we have the flexibility to choose the optimum routing (and data aggregation) functions for each application.

Security features are also added as optional services. Examples of this are authentication and encryption protocols. Some security features are best supported by the hardware, particularly where some protection against malicious attacks is required. Some of the issues that apply specially to PEN are discussed in [Stajano et al 99].

Protocols Conclusion

The protocols outlined above do allow for some complexity in nodes that can support it, while remaining interoperable with nodes that have made the simplest choices, and may not even require a processor. As much of the design as possible has been cast as hints, to be included and used only by more capable nodes. It is envisaged that they will be used by some applications that need to use the limited bandwidth more efficiently, or with lower latency, or with lower power consumption. It is our goal however to leave enough of the channel free for simple nodes with little computational power to use it effectively.

Spectrum Pricing

Is it realistic to assume that other users of the channel are cooperating in order to save power? How much would it cost to provide a separate radio band for PEN services? A simplistic analysis can be made relative to the recent auction of spectrum in the UK for 3rd generation (3G) mobile devices. The licenses to use this 145Mhz of spectrum for the next 20 years were sold for a total of 35 billion dollars. For more details, see the website http://www.spectrumauctions.gov.uk/3gindex.htm.

This prices spectrum at 12 dollars per hertz per year to cover the UK. To purchase 100kHz for 20 years would cost 24 million dollars. As the population of the UK is about 60 million people, this is 40 cents per person, or about 2 cents per person per year. For comparison, the 3G licenses have cost 29 dollars per person per year, or about 1500 times more, reflecting the 1500 times more bandwidth required. If we allow a factor of ten for the lower frequency and smaller size of this band, then we arrive at 20 cents per person per year which should still not be an insurmountable hurdle.

Perhaps what this points to is that for a reasonably sized business (alarms, heating controls etc.) it could be possible to purchase a spectrum allocation in which the protocols can be closely controlled, without increasing the cost significantly.

Prototype Applications

To test the PEN protocols under real conditions, we have built a range of hardware devices.

HARDWARE FEATURES	APPLICATIONS
No connections	Beacon Logger
Serial connection	Download station General PC/internet connectivity
User Input	Button box Cyberspice case
Relays	Fan Light switch

Sensors	Thermometer GPS Light detector Camera
User Output	Display

The simplest hardware configurations are where the nodes do not require any sensor or communication facilities. Examples of this are loggers or beacons. Beacons simply supply pre-programmed information to other nodes, such as location, time, inventory or time zone. Passing nodes can use this information to record where they have been and when, or may be authorised to update the information. Loggers collect information from designated nodes, and download it to others. Whenever they contact a node containing relevant information, they request copies of that information, avoiding duplication, and download it.

Another class of hardware device is a switch. Our simplest switch application consists of a CD case whose spine contains a magnetic switch and PEN node, and a track advance button on the inside of the case. We called this system “cyberspice”, after an earlier prototype built into a spice jar, and a pop group that were popular at the time (see figure 2).

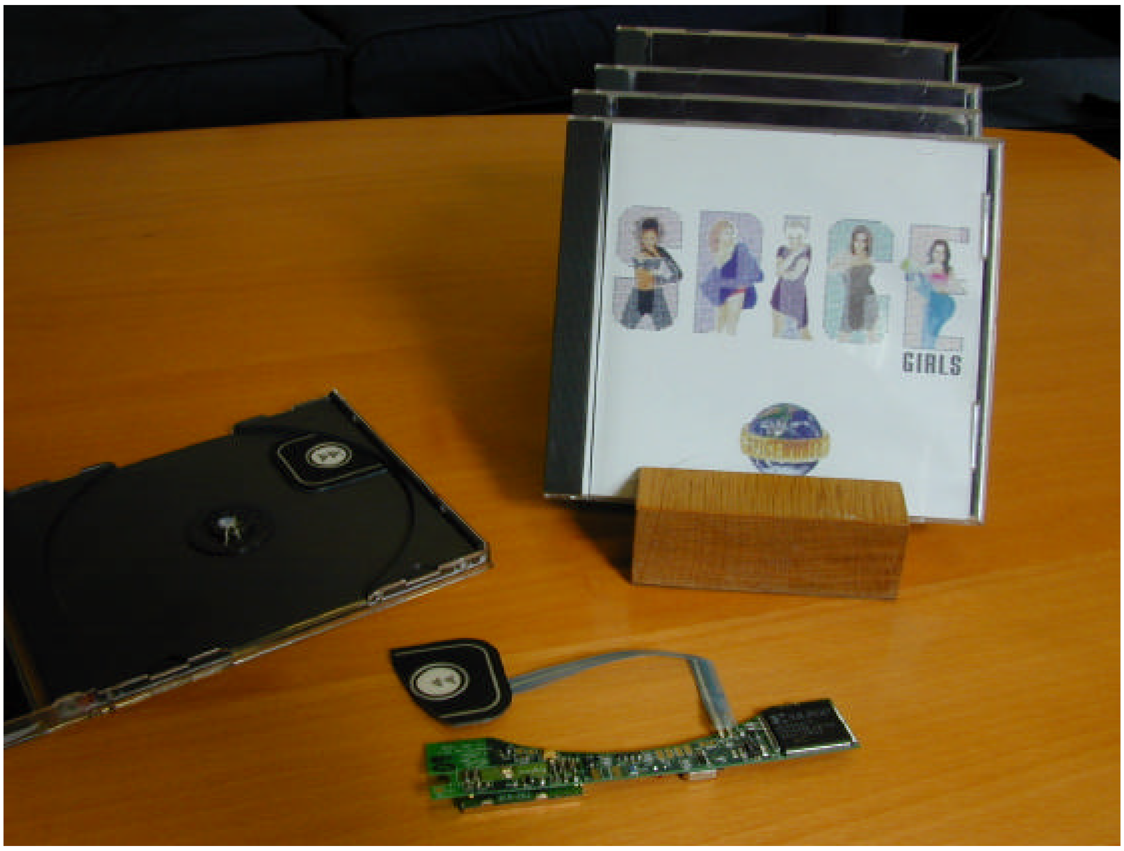

Figure 2. Cyberspice nodes with transmitter PCB.

The transmissions encode information on which case has just been opened or closed, whether the case is still open, and if the track advance button has been pressed. The power saving on these nodes is particularly easy. We can power down completely until the case is opened, which causes the magnetic switch to close, and initiates the transmissions.

The CD case is used to control a PC-based music player. Many empty CD cases are equipped with these nodes, and the player is programmed to play the appropriate tracks whenever one is opened nearby. In the first version of the CD case, we only use a transmitter. We blindly repeat the transmission a few times when the case is first opened so that there is a reasonable probability of one of them getting through. Other traffic could be disrupted by this, but all of our MAC protocols assume packet loss, and should recover immediately. The case then transmits its ID once a second while it remains open, so that the music player knows that it is still around, and can stop the music by a timeout if no transmissions are heard for three of those intervals. When the case is closed, we transmit the identity and a flag to say the case is closed, and repeat that several times to ensure that at least one gets through. Thus in the common situation of the case being in good contact with the player, the music starts and stops within a few tens of milliseconds, but if the case wanders out of range, the music stops anyway after about a few seconds, and restarts a few seconds after it re-enters radio contact.

We have also built a slide advance box whose buttons are used to control presentations in our meeting room. The transmissions consist of the unique ID of the box, and information on which of two buttons has just been pressed. This button box was designed after cyberspice, and uses the full PEN hardware and MAC layer. When switched on, the box uses a multicast to contact any listening nodes. The multicast contains a request for any service that is looking for buttons to control it. If a node in the vicinity is running such a service, it will reply to indicate that it is interested in being controlled by that box. The box then sleeps until a button is pressed, at which point the MAC layer is invoked to send a unicast packet to the button controlled service identifying the box and the button event, with a retry count of five. This will result in the node listening for a free channel, transmitting the information, and then listening for an immediate acknowledgement. If this is not received, the node will sleep for a short backoff interval, and then attempt the transmission again. It will repeat this cycle up to five times, then give up and leave it to the user to press the button again.

Simple sensors such as thermometers spend most of the time powered down awaiting a wakeup signal from an on-board timer. They then sample the sensor, store the results if necessary, and sleep again. They beacon occasionally in order to rendezvous with cache or logger nodes. A cache node is a listening node, often with a bigger battery than the sensors, or plugged into a power supply. It gathers the readings from any sensor nodes in its vicinity in small batches, and when it sees the beacon from a logger or download node it provides them with the latest readings if they have not already found them. In this way, it saves the logger node having to spend time listening for sensor nodes. A logger can also upload data from sensors directly when one of its buttons is pressed. It listens for a couple of beacon intervals to gather data on which sensors may be around, and then contacts them for their latest readings. A download node is attached to a controlling PC by its serial port. It listens for the beacons of logger or cache nodes, and requests their latest data if it does not already have it. It then transfers this data to a waiting application on the PC, where it is made available for display, analysis and dissemination.

We have made a basic kit of about eight temperature sensors, caches, loggers and download nodes for use in diagnosing temperature problems. The first surprise when distributing the nodes around a house was that there was a sufficiently good channel from every sensor, through caches and loggers, to the download node, that the logger node did not have to be carried around to fetch the readings. It was also noticeable that after a long period, around an hour, a logger and sensor could accidentally overhear each other's beacons while listening for a free channels for their own, and make a long term rendezvous without entering a listening phase. Once this had happened, the data started to flow through the logger to the download node as soon as each batch of it had been collected, so we were getting data displayed after a delay of just a few minutes. With better radios, we will not be listening for a free channel for so long, so these accidental rendezvous will be less likely. We will then have to rely on the original scheme of pressing a button, or having higher-powered cache or proximity store nodes, to make the introduction between sensor and logger.

Examples of the temperature charts for five sensors are shown in figure 3.

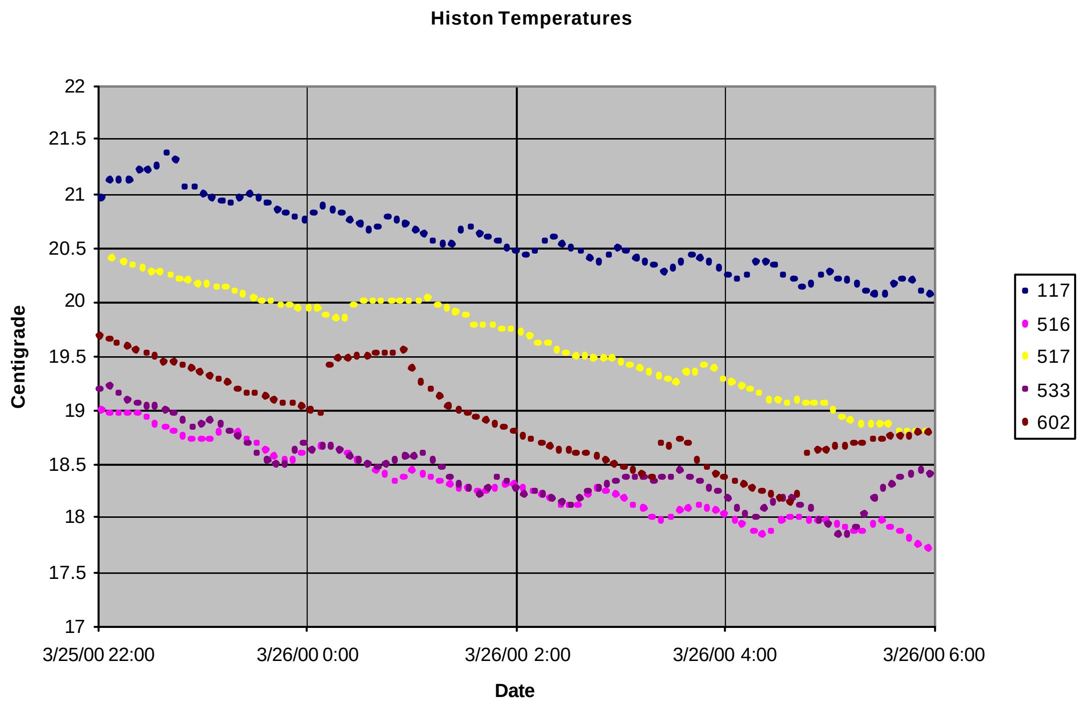

Figure 3. Temperature Charts

Temperatures have been taken every five minutes during the night, and it can be seen that the central heating has been coming on for about 15 minutes every 45 minutes, but that the general temperature of the house declines during the night.

The temperature nodes have now been distributed around our new machine room to help solve problems with the chilled airflow, which is not reaching all the machines that need it, and is wasting energy. The PEN nodes are considerably smaller than the mechanical recorders that they replace, so they do not disturb the airflow so much (see figure 4).

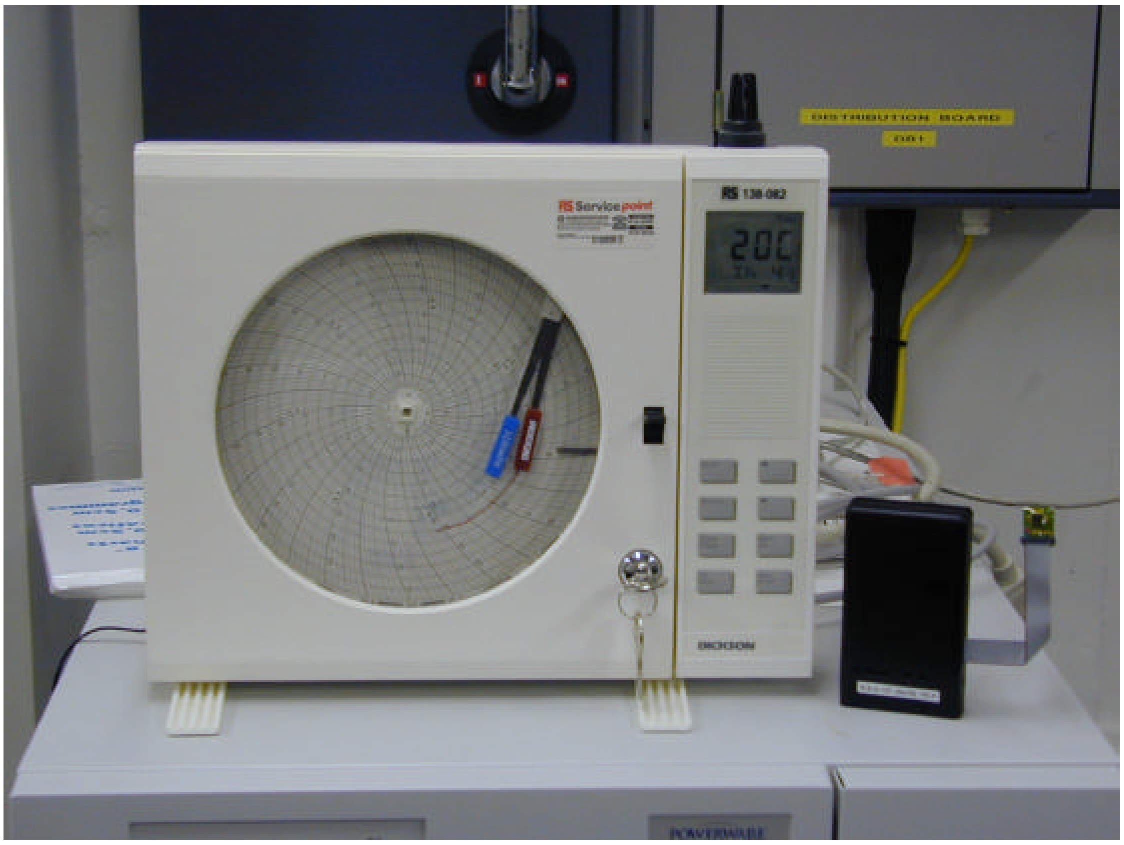

Figure 4. Temperature chart recorder alongside a PEN temperature sensor.

They are cheap enough for more of them to be deployed, and they send their readings directly to a server that makes them available in standard file formats and as a web page.

Amongst other simple applications built to test PEN were a light controlled by a remote light sensor and a fan controlled by a remote thermometer (see figure 5).

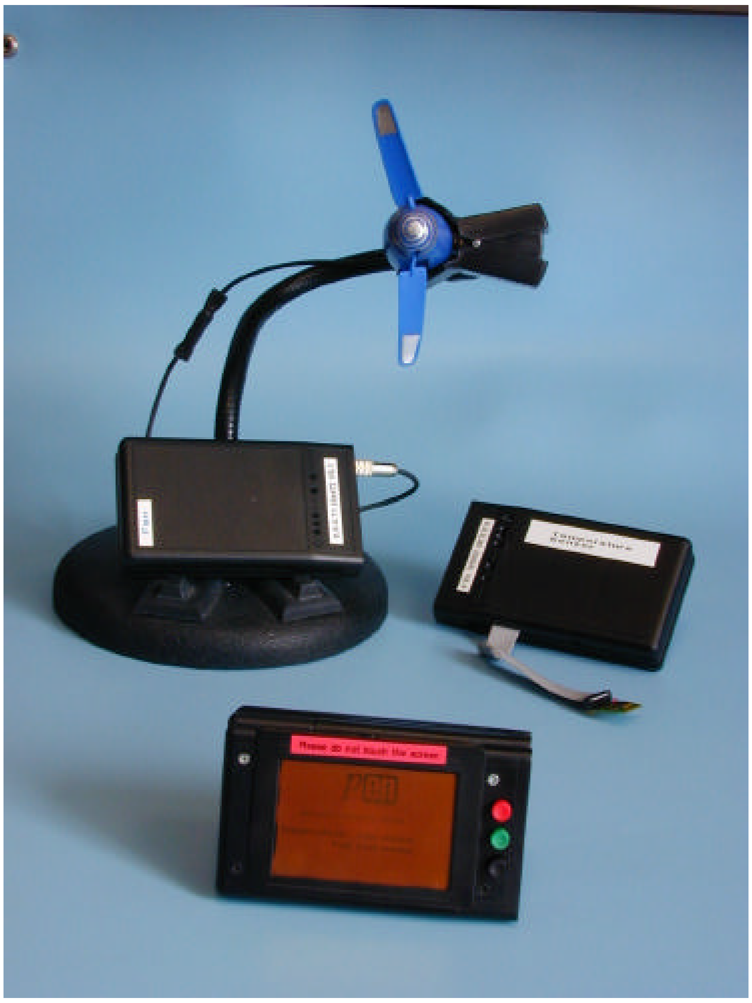

Figure 5. Fan, thermometer and display.

More complex nodes included a VLSI camera module whose images were compressed in such a way as to preserve the edges required for target recognition of a 2-D circular barcode. The steps of thresholding to a 1 bit image combined with edge extraction were simple and fast, leaving only a tiny dataset which was then transmitted to a server machine. The server ran the computationally intensive task of finding the arcs of ellipses that outlined the bit codes. The viewing of a target by the wireless camera could then be interpreted as a door being open, a particular page being visible on a desk, or a car parking space being vacant. The camera could also have been programmed to look at general light levels, changes in the scene, or to detect the motion of the camera.

A GPS node was built where a GPS receiver module was combined with a PEN node and rechargeable batteries. This could be placed in a car, bicycle or any object with a clear view of the sky, and would record its position every few seconds. When a logger node passed by, it would download the latest readings to the logger, which would then transfer them to any suitably configured download node on a PC for analysis, in much the same way as the temperature sensors.

Finally, some display nodes were built with non-volatile displays, so that they could be battery powered user interface devices. These were used to configure and observe some of the applications described above, but also in their own right as electronic door signs connected by PEN to our intranet. They were programmed to show visitors which way to go for their next appointment, to display urgent messages, and to describe what was currently behind the door (see figure 6).

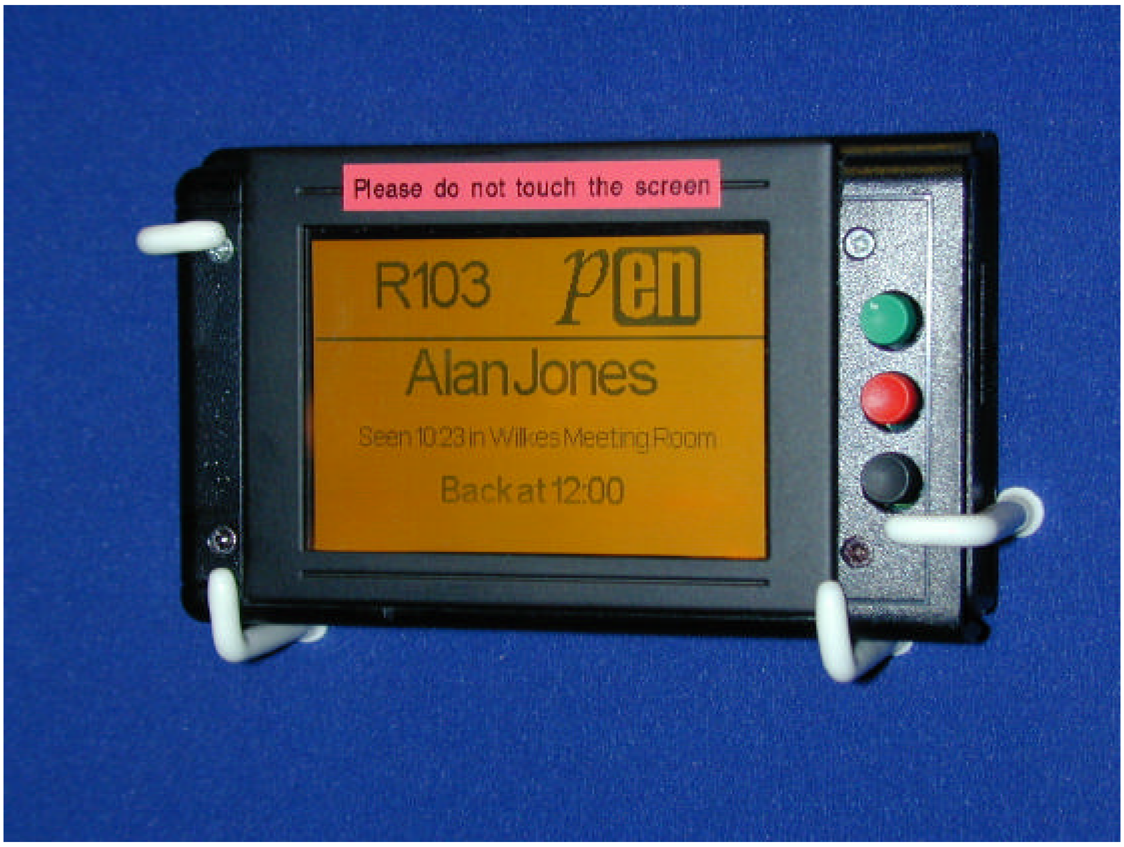
Please do not touch the screen
R103
pen
AlanJones
Seen 10:23 in Wilkes Meeting Room
Back at 12:00

Figure 6. PEN display node acting as a door sign.

Conclusion and Future Work

We have presented here some of the design decisions that went into the PEN project, and tried to compare them with alternative approaches.

We have designed all aspects of the system looking forward to a better radio implementation. The new radios that we are designing will have faster wakeup from sleep to receive, much shorter turnaround times from receive to transmit, particularly crucial where there is contention for the channel, and some indication of received signal strength that can be used when judging whether or not to transmit over another packet.

The choices presented for beaconing protocols each have their place in different application areas, and we hope that the overheads of making them interoperable will be low enough that all nodes can communicate at that level.

The lack of any form of group synchronisation in PEN allows us to incorporate very simple nodes, but does limit the channel utilisation that can be achieved. It makes it harder to guarantee channel capacity for special cases such as audio streaming and priority alarm channels. These sorts of traffic can be handled "best effort" within PEN with limits on how much of the channel they can utilise, but we would not want to compromise its simplicity and tiny bandwidth requirements for the sake of one or two traffic patterns that are already handled well by existing systems.

Our work is now focusing on the integration of the radio and digital components to achieve the power, size and cost factors that make wide scale deployment possible. Only then can we judge the effectiveness of the protocols and the usefulness of the new applications that they enable.

References

[Bennett et al 97] Bennett F, Clarke D, Evans J B, Hopper A, Jones A, Leask D,
"Piconet – Embedded Mobile Networking", *IEEE Personal Communications*, Vol 4 No 5, Oct 1997, pp 8-15.

[Bluetooth 99]
Bluetooth Specification Volume 1, Bluetooth Consortium, July 1999

[Girling et al 00] Gray Girling, Jennifer Li Kam Wa, Paul Osborn, Radina Stefanova.
"The Design and Implementation of a Low Power Ad Hoc Protocol Stack".
Proceedings of IEEE Wireless Communications and Networking Conference 2000, September 23-28, 2000, Chicago, IL, USA.

[Stajano et al 99] Frank Stajano and Ross Anderson.
"The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networks", *Security Protocols, 7th International Workshop Proceedings*, Lecture Notes in Computer Science. Springer-Verlag, 1999. http://www.cl.cam.ac.uk/~fms27/duckling/.
Also available as AT&T Laboratories Cambridge Technical Report 1999.2.

[Todd et al 00] Todd T D, Bennett F and Jones A,
"Low Power Rendezvous in Embedded Wireless Networks", *Proceedings of IEEE First Annual Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC 2000),* August 11, 2000, Boston, MA, USA.

[Weatherall et al 99] James Weatherall and Andrew Hopper
"Predator: A Distributed Location Service and Example Applications", *Proceedings of Cooperative Buildings 1999 (CoBuild'99),* Springer Verlag Lecture Notes in Computer Science.