A more recent version of this paper appears in IEEE Micro, February 1996. An older version also appeared in the IEEE Hot Interconnects Symposium ‘95, August 1995.

Experiences with a Wireless Network in MosquitoNet

Stuart Cheshire and Mary Baker

Computer Science Department, Stanford University
Stanford, CA 94305-2140
{cheshire,mgbaker}@cs.stanford.edu

Abstract

As the trend towards smaller, lighter and more powerful computers continues, mobility and wireless connectivity become increasingly important.

The MosquitoNet project is studying issues in wireless and mobile computing, and this paper presents our performance analysis and observations of one of the emerging wireless technologies: the Metricom Microcellular Digital Network (MCDN).

Metricom provides one of the best combinations of bandwidth, coverage area and cost for a wireless data network. We find that the throughput of their radio devices is comparable to a modern modem, but the latency, or round-trip delay, is much higher. The maximum throughput we measured was 30-40 Kbits/second, but the minimum latency for even the smallest IP packet is at least 60ms.

We have experimented with both the datagram and Hayes modem emulation modes of Metricom’s wireless radios. We are early users of the datagram mode, and our comparisons of these modes confirm two well-known but sometimes forgotten principles. First, packet switching allows more efficient sharing of resources than does circuit switching. Second, interfaces that are satisfactory for use by human beings often show their flaws, ambiguities and omissions when used as programming interfaces for software control of devices.

1. Introduction

With smaller, lighter, more powerful computers becoming ever more common, the MosquitoNet project [1] is exploring how to keep these computers permanently connected to the world-wide Internet. This connectivity is both wired and wireless, depending on physical location and the availability of wired access points. We anticipate that the number of wireless services and their coverage areas will increase to make ubiquitous connectivity possible. Eventually we hope people will be able to rely on having access to the same information and services on their portable computers that they have on their desktop workstations.

To support this vision of ubiquitous connectivity, we are studying issues of wireless and mobile computing at the network, system and application levels. Our two main goals in support of the MosquitoNet vision are to provide seemingly continuous connectivity for mobile hosts, and to provide support for system- and application-level adaptation to dynamically changing network characteristics. We are investigating what changes and extensions to standard networking APIs are needed to allow a well-written application to deal gracefully with widely varying network characteristics, especially when the service characteristics can change without warning at any time, even while the application is running.

To execute our work in supporting adaptive software, we are developing a mobile computing testbed. This consists of two components: first, a Mobile IP implementation (based on the IETF’s Mobile IP specification [6]) to allow hosts to switch between different network interfaces, and second, drivers and support for several different network technologies. These two components are important, because ubiquitous connectivity does not imply that the entire world should use the same physical network technology; rather, it implies that computing devices should be able to switch between many different physical layer technologies in order to use the best that is locally available.

The current hardware platform for our test-bed consists of industry-standard x86 personal computers running Linux [2][5][9], a free UNIX-like operating system. Our desktop machines, which serve as IP routers connecting between the wired and wireless networks, are PCI-bus PCs with 90MHz Pentium processors. Our mobile computers also run Linux and are Gateway 2000 'Handbook' computers that weigh under three pounds and have 40MHz 486 processors and 20Mbytes of RAM. We describe the network technologies used in our test-bed in the following subsection.

1.1 Network Technologies

One basic requirement for the MosquitoNet project is a range of network technologies for the system to choose between. For our initial implementation, we chose two technologies, one wired, and one wireless.

We chose 10Mb/s Ethernet as our wired technology, because it is cheap, available in a wide range of form factors, and is widely supported.

For the wireless technology, we needed a service with characteristics significantly different from Ethernet while still being usable for a data network. We chose radio devices made by Metricom, Inc. of Los Gatos, California. Metricom's is currently the only affordable service that covers most of our metropolitan area and offers performance roughly comparable to a good modem. And Metricom's service, unlike cellular telephone-based services, does not charge a per-minute or per-packet usage charge.

In the future we intend to support other types of network devices from the range shown in Table 1. We hope eventually to support as many of the appropriate technologies as is practical, but for our initial implementation we chose the two technologies with the best cost/benefit ratio.

We considered 'Wireless Ethernet' products, but their performance does not provide as dramatic a contrast to the wired Ethernet. They have high bandwidths and low latency compared to the wider-area wireless technologies, and due to their short range they are usually limited to in-building use. Since the eventual goal of the MosquitoNet project is continuous network connectivity everywhere, our first wireless technology needs to be something with longer range. Metricom's radios provide a better approximation to the ubiquitous connectivity that is our goal.

Technology	Throughput	Latency	Characteristics
100Mb/s Ethernet	10^8 bps	< 1ms	Wired
10Mb/s Ethernet	10^7 bps	< 1ms	Wired
Wireless LAN	10^6 bps	1-10ms	Short range
Wireless MAN	10^5 bps	≥ 100ms	Slow
Wired modem	10^4 bps	≅ 50ms	Slow, wired
Cellular modem	10^4 bps	≅ 50ms	Slow, expensive
CDPD	10^4 bps	≅ 500ms	Slow, expensive
Pager	10^3 bps	seconds	Slow, expensive

TABLE 1. Available network technologies.
This table summarizes some of the available network technologies, ranked by throughput in terms of bits per second (bps).

We also considered CDPD (Cellular Digital Packet Data) but at the current per-packet pricing rates of $80 per megabyte it works out to be even more expensive than the 40¢ per minute connect time cost of a cellular modem.

Pager services such as ARDIS also provide wide area coverage, but at the current data rates and pricing they are unsuitable for use as a technology to provide a full-time Internet connection.

1.2 Overview of Results

This paper presents our measurements of the throughput and latency of communication over the Metricom radios. The radios have a transmission rate over the air of 100 Kbits/second, but in our tests we achieve at best one third of that due to software overhead in the radios themselves.

The latency for a single IP packet is 60ms fixed overhead + 420µs per byte. For example, the time to deliver a 100-byte packet is 102ms, and for a 1-Kbyte packet it is 490ms.

When sending a continuous stream of packets, we find some pipelining benefit. The incremental cost for each packet after the first is reduced to 25ms + 260µs per byte, meaning that the radios can deliver one 1-Kbyte packet every 290ms, giving a maximum throughput of roughly 30 Kbits/second.

We find that using the radios in their packet-oriented mode of operation (which Metricom calls 'Starmode') yields more consistent results than using them in their modem emulation mode. Although the throughput is comparable in both modes, the modem emulation mode frequently suffers long delays in our tests.

We also found that the programming interface for the radios is designed with a human operator in mind. When writing software to drive the radios, certain aspects of the interface are awkward to deal with, including initialization overhead and detecting error conditions.

In the rest of this paper we describe our experiments and results in more detail. The next section describes Metricom's Microcellular Digital Network service and Section 3 describes our implementation of IP over Metricom's hardware. Section 4 describes the tests we performed and the performance results we obtained. Section 5 examines some issues in hardware/software interface design, and Section 6 presents our conclusions.

2. Metricom's Service

In this section we describe the technology that makes up Metricom's service: the radios, the design of the wide area network, and the way that the portable radios interface to the mobile computers that use them.

2.1 Underlying Technology

Metricom's radios operate in the 902-928MHz band allocated by the FCC for use by unlicensed low-power devices. The radios use frequency hopping, jumping between 160 distinct channels in that range, and can achieve an over-the-air transmission rate of 100 Kbits/second.

Because each of the radios continually changes its receiving frequency, any radio wishing to communicate with another needs some mechanism to determine what frequency it should use. The sequence of frequencies used by each radio is determined by a pseudo-random number generator. Every 25ms the radio retunes its receiver to the channel dictated by the next number in the pseudo-random sequence, changing channels 40 times per second.

Each radio also devotes a small percentage of its time to discovering its neighboring radios. It maintains a list of its neighbors, along with information about their current position in the pseudo-random channel sequence. When a radio wishes to transmit a packet to one of its neighbors, this stored information allows it to predict what channel that radio will be listening on at a given moment.

Once transmission of a packet starts, the radios remain on that channel and suspend their hopping until the transmission is complete. However, the pseudo-random number generator continues to run at the prescribed rate, so that the timing and sequence information held by other neighboring radios will remain correct.

Metricom's radios are packet-oriented, not circuit- or call-oriented, and this allows network performance to degrade gracefully when overloaded. New channels are selected on a packet-by-packet basis instead of on a call-by-call basis. Thus there is no way for idle circuits to tie up channels as can happen in the cellular telephone network.

Due in part to the packet nature of their network, Metricom charges a flat monthly fee, regardless of 'connect time.' Metricom does not need to construct their pricing model to discourage people from staying connected for long periods of time, because they have a technology that supports fine-grained sharing and makes efficient use of the available bandwidth instead of dedicating a channel per user.

2.2 Network Infrastructure

Metricom's wide-area network infrastructure consists of fixed outdoor radios that are usually mounted on top of street lighting poles. This gives the radios both reasonable height off the ground and a convenient source of electrical power. The pole-top radios communicate with each other to manage the network and provide directory services.

The network uses geographical routing. At installation time these 'pole-top' radios are told their precise latitude and longitude using a hand-held GPS receiver. The pole-top radios are spaced roughly half a mile apart and act as repeaters: they forward packets from one to another to reach geographically-determined destinations. Instead of using manually (or automatically) configured routing tables, such as those used by an IP router, a pole-top simply compares the latitude and longitude of the packet's destination with its own, and gets the packet one hop closer to its final destination by forwarding it to the best reachable pole-top in the right direction.

Roughly 10% of the pole-top radios also have a wired connection. If a packet has to go a long way, a pole-top may elect to deliver it to the wired network to save sending it through too many wireless hops. Through these wired access points, the Metricom network can also route packets onto destination wired networks.

2.3 Portable Radios

Mobile connectivity is provided by small, battery-operated portable radios. These radios connect to the computer via an RS232 serial port, and support rates up to 115,200 bits/second. They operate in almost exactly the same way as the pole-top radios, except that they are not configured to function as repeaters.

Because the portable radios have no fixed geographical location, they require some mechanism to locate each other. Each portable radio registers with its closest pole-top radio so that its location may be known to any other radios that are out of range of direct communication. Packets destined for the portable need only be delivered to the closest pole-top, which then forwards the packet to the portable.

Because the radios maintain lists of other radios within range of direct communication, portable radios close to each other may talk to each other directly. This makes it possible to take a group of Metricom-equipped laptop computers to an area that does not have the Metricom pole-top service in place, and still have them communicate with each other. In contrast, if you take a pair of cellular telephones (or CDPD modems) to Yosemite, they are completely useless, because there is no cellular base station there to make them work.

2.4 Host Interface

The Metricom radios can operate in two distinct modes. They can be used to emulate a Hayes Modem, setting up point-to-point connections with the usual Hayes 'AT' command set, or they can be used in "Starmode," directly sending and receiving individually addressed packets.

When using the radios in the Hayes Modem emulation mode, users can connect to other radios, via a PPP server to the Internet, or via a wired gateway to other conventional wired modem services like America Online or Compuserve. The radios set up a TCP connection over the underlying packet-switched wireless network to provide a reliable byte-steam connection to emulate the modem 'call.' Metricom's motivation for this support is that it allows users simply to drop in a set of Metricom radios in place of standard Hayes modems and use the radios with little, if any, change to their software.

For our purposes Starmode is the more interesting mode, because Starmode packets can be individually addressed to specific destinations without any prior connection setup. In this mode the radio behaves much more like a true network interface. Because it is datagram oriented, there is no fixed limit on the number of simultaneous end-to-end 'connections' a host can maintain.

A benefit of packet-oriented communication is that not only can mobile clients be in simultaneous communication with any number of other mobile hosts, but so can non-mobile wired Internet hosts. In the MosquitoNet project one of our desktop computers acts as a router, connecting our wireless subnet to the rest of the Internet. To do this, the router only needs one Metricom radio connected to it, because we use the radio in packet mode, allowing the router to remain in communication with any number of mobile hosts simultaneously.

In contrast, if we used the radios (as currently designed) as modems, we would need to purchase an entire 'modem bank' of radios to connect to our router. We would need one modem per active client, just as dial-up SLIP servers do. The widespread use of analog modems today is a result of the low cost and high availability of telephone lines, and not because telephone lines are the ideal way to connect computers together. With new wireless technologies, we have the opportunity to design our system correctly, not merely to mimic existing sub-optimal solutions.

2.5 Using Starmode

The format of a Starmode packet is very simple, as shown in Figure 1. The address field of the packet is denoted by an asterisk before and after. The payload of the packet can contain any data. We adopt the convention of beginning the payload with a characteristic four-character code, so that we can easily distinguish our packets from other unrelated Starmode traffic we might receive. In this sense, the field functions like the protocol ID field of an Ethernet packet. The End of Packet marker is a carriage return character, which means that we must use a byte-stuffing algorithm to eliminate this byte value from the payload.

The address field is the most curious part of a Starmode packet, because it has a different meaning depending upon whether the host is sending or receiving a packet. The address field contains the "name" of a radio, in ASCII text. Each radio has a permanent name, usually a pair of four-digit numbers, as shown. (Additional names may also be assigned if a more descriptive identification is desired.) When sending a packet, the address field contains the destination radio's address. When receiving, it contains the source radio's address. This is because Starmode is the programming interface by which a host communicates with the radio over the serial port, rather than the actual 'over-the-air' packet format used by the radios.

*	0000-1164	*	SRIP............	CR

FIGURE 1. Starmode packet format.
This figure shows the format of a Starmode packet. The Start of Address (SOA) and End of Address (EOA) are marked by asterisk characters. The address field contains the destination of the packet when sending, and the source of the packet when receiving. The End of Packet (EOP) is marked by a carriage return. The payload begins with four characters (SRIP) that distinguish our protocol's packets from others.

3. Protocol Implementation

In order to use existing IP applications with the Metricom radios it is necessary to encapsulate IP packets using the Starmode interface. Our protocol is called STRIP, which stands for STarmode Radio IP. STRIP uses a straightforward encapsulation scheme that simply sends the IP packet as the payload of a Starmode packet addressed to the correct destination, much as is done with Ethernet.

Since the Metricom radios communicate with the host over the serial port, we based our driver code on existing SLIP code. We added code to look up the radio address for a given IP address and prepend a Starmode header to the IP packet. This is necessary, because a SLIP driver does not normally include any addressing code, since it correctly assumes that there can only be a single host at the other end of the serial line.

Mapping IP addresses to the correct link-layer (radio) addresses is the most difficult issue for the STRIP implementation. Ironically, despite being radio devices, the Metricom radios have no broadcast capability, making a solution like the Ethernet ARP protocol inappropriate. Due to their independent channel-hopping operation, designed to minimize interference between simultaneous transmissions to different radios, there is currently no way for a single transmission to be received by all radios simultaneously. In our current software, the address translation tables are manually administered. However, we are now planning to implement an automatic directory service incorporated into a DHCP [4] server that manages dynamic assignment of addresses.

4. Performance Measurements

This section presents throughput and latency measurements for IP traffic over the Metricom radio interfaces. We were interested in seeing how much of the 100 Kbits/second air transmission time we could actually achieve. Our tests show that STRIP throughput is at most 32 Kbits/second and is limited by overhead inside the radio firmware. We measure the time for both single packets and bursts of packets to determine the possible benefits of pipelining. The second and subsequent packets in a burst of packets incur less overhead than the first. We also find that per-packet latencies are very high – at least 60ms for even the smallest packets. This includes the overhead of the serial interface on the sending and receiving radios. For comparison, we also measure transmission times using the radios' modem emulation mode. Modem emulation throughput is marginally better, but with much higher variance and much lower worst-case throughput.

For all of our tests, we measure one-way transmission delay, rather than the normal round-trip delay measured by the UNIX 'ping' command. Since the radios have only one antenna, they cannot send and receive simultaneously and are thus half-duplex devices. We measure the one-way transmission delay to avoid any interference between outgoing and incoming traffic that might make our results more difficult to interpret. However, this means that our results may not be achievable for applications that really require full-duplex communication.

In order to measure the one-way transmission delay, a dæmon on the receiving side sends back a small acknowledgment over the Ethernet for each packet it receives. The transmission delay over the Ethernet (1ms) is insignificant compared to the delay over the wireless interface, making this an effective way to measure the one-way transmission times.

All of these tests were performed under good conditions. Packets required only one hop to travel between the source and destination radios, and we believe there was no other traffic within range during our tests.

4.1 IP Packets over Starmode

We first measure the time it takes to send individual packets of varying sizes. This gives us both throughput and latency measurements, including the overhead of the serial interfaces and the 'airtime' for the data. Figure 2 shows the minimum, average and maximum one-way transmission delay measured (under good conditions) for IP packets varying in size from 64 bytes to 1 Kbytes. The Metricom header is not included in the packet size, since it is unavoidable fixed overhead for any data sent over the radios. For each packet size, our tests send 64 separate packets. For each group of packets, we record the minimum, average and maximum delivery time.

We see that the best line intercepts the time axis at 60ms, and has a gradient of 420µs/byte. This tells us that the time to send a packet from one radio to another is at least 60ms fixed overhead plus an additional 420µs/byte. This translates to a maximum throughput of 2380 bytes/second or 19 Kbits/

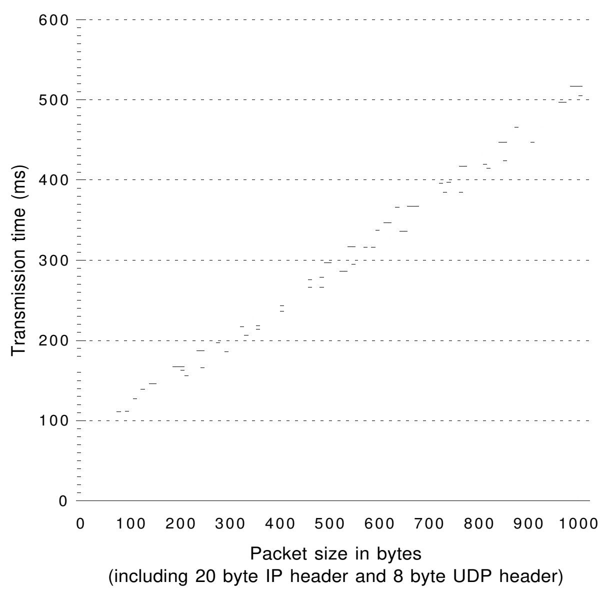


FIGURE 2. STRIP transmission times

This figure shows the minimum, average, and maximum one-way transmission times for STRIP packets varying in size from 64 bytes to 1 Kbytes. The packet size is the entire IP packet; the size of the Metricom header is not included. At each data point we performed 64 one-packet tests. The lowest line shows the best time for the given size. The middle line shows the average time at that size, and the top line shows the worst time.

second, a long way short of the 100 Kbits/second air transmission speed.

To determine the reason for this low throughput, we analyze the component times for transmitting a single packet. We find that part of the reason for this low throughput is the latency of the serial interfaces, and part of the reason is overhead in the radio firmware.

Figure 3 pictorially shows the component times that make up the total transmission time for a packet sent between radios. Because the radio is connected via a serial port, it takes a non-trivial amount of time for the host to deliver the packet to the radio, and for the receiving radio to deliver the packet to its host. The time to send a byte over a serial port at 115200bps is 87μs (1 start bit + 8 data bits + 1 stop bit = ten bits per byte). Given the 60ms constant overhead and a 87μs/byte serial delay for sender and receiver, the wireless transmission time must be 60ms (fixed overhead) + 420μs/byte - (2 * 87μs/byte) = 60ms + 246μs/byte.

Even assuming that the fixed overhead could be amortized over an arbitrarily long transmission, 246μs/byte translates to a maximum throughput over the air of 4058 bytes/second, or 32 Kbits/second. This still a long way short of Metricom's air transmission speed of 100 Kbits/second. We have performed further isolated tests that indicate the overhead is not in our software but in the radio firmware for supporting Starmode. Metricom confirms this and is working to address this issue.

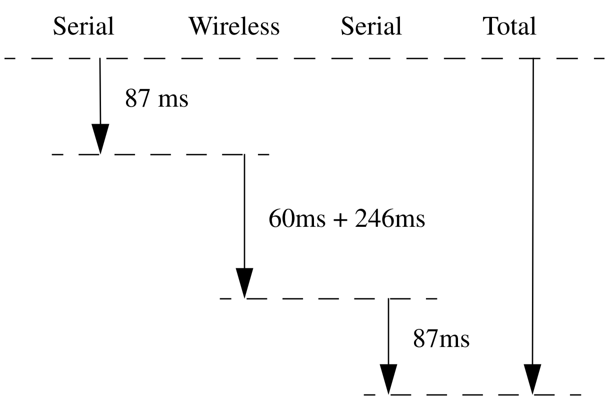


FIGURE 3. Components of packet latency.
The total time to send a 1000-byte packet from one radio and receive it on another includes a serial port delay to deliver the packet to the radio, wireless transmission time (or 'airtime'), and a second serial port delay to deliver the packet to the host at the receiving end. The times shown are for a 115200bps serial interface.

4.2 Packet Pipelining in STRIP

Since the high packet transmission latency is partly due to the three stages of the pipeline operating serially, we measure the effect of sending a stream of packets from one radio to another. This should allow all three stages of the pipeline to operate in parallel, increasing the throughput. The total time for delivery of any particular packet remains the same, but due to the potential parallelism, the rate at which the radio delivers the packets is improved.

Our measurements show that the time to send two packets is indeed less than twice the time to send a single packet. Figure 4 displays the transmission times for each packet in a burst of eight packets. It shows that the cost to send one 1000-byte packet is about 500ms, but each additional 1000-byte packet adds only 300ms to the total time. In order to illus-

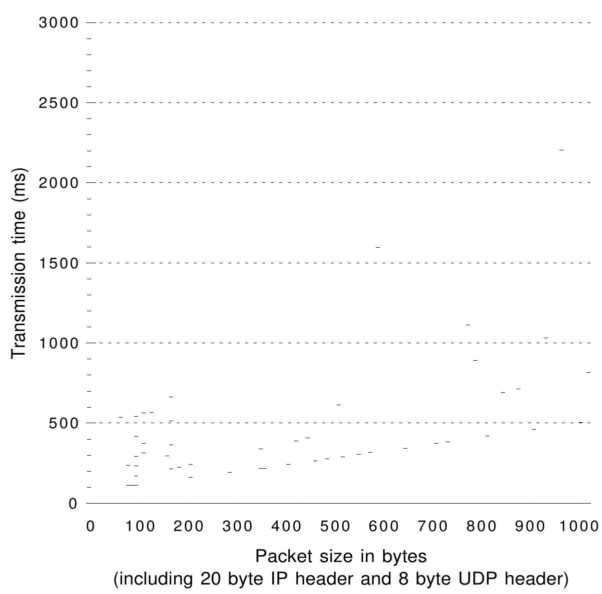


FIGURE 4. Packet-burst transmission times.
This figure shows the one-way transmission times for bursts of packets varying from 64 bytes to 1 Kbytes. For each packet size, we sent a burst of eight packets. The lowest line for each packet size shows the transmission time for one packet. The next line shows the transmission time for two packets, and so on. The top line shows the transmission time for the entire burst of eight packets.

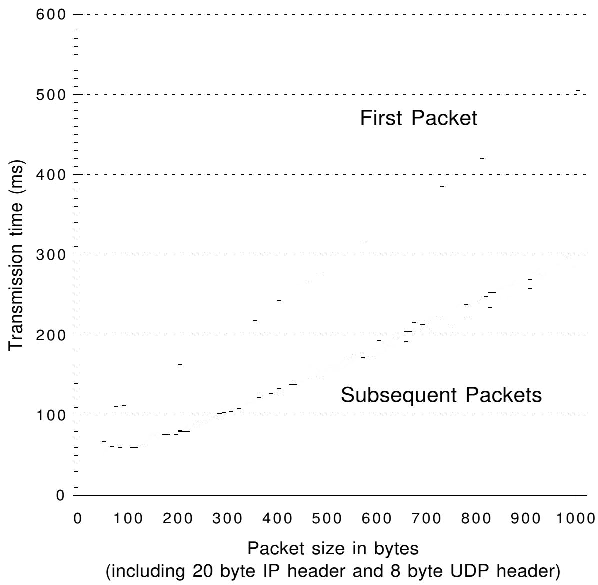
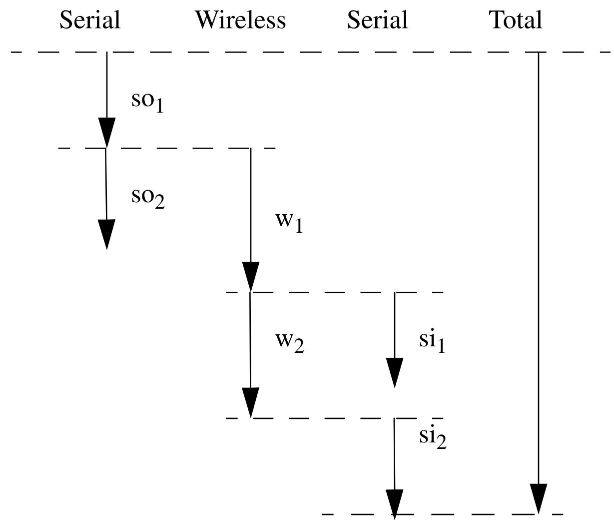


FIGURE 5. Incremental packet burst delays. This figure shows the cost for the first packet in a packet burst and the incremental costs for the subsequent packets in that burst of packets. As in the previous figure, each packet burst consists of eight packets of a given size. The top line shows the cost of sending the first packet. The lower lines show the additional cost for sending each subsequent packet as compared to the total cost for the previous packets.



$\text{Total} = so_1 + w_1 + w_2 + si_2$

FIGURE 6. Pipeline timing diagram. This figure is a timing diagram for a pair of pipelined packets. The out-going serial delay for the second packet (so_2) is hidden by the wireless transmission time of the first packet (w_1). Likewise, the in-coming serial delay of the first packet (si_1) is hidden by the wireless transmission time of the second packet (w_2). Because of this, the total transmission time only includes two serial delays for the two packets, rather than four.

trate this observation more clearly, Figure 5 shows the incremental cost, or additional cost, for each packet in the burst of packets.

That we see some benefit from pipelining is not surprising, since this phenomenon occurs in many other networks. For instance, the reason that TCP/IP works better than Novell Netware over long distances (links with a high delay-bandwidth product, such as satellite links) is because TCP/IP is a variable-window protocol and Novell Netware is a one-packet stop-and-wait protocol.

What is somewhat surprising is that we do not see any additional pipelining benefit after the second packet. One might expect that to get the full concurrency from a pipeline with three stages, it would take at least three packets.

Figure 6 illustrates why the second packet realizes the full benefit of the pipeline. On the outgoing serial port, the transmission time of the second packet is masked by the wireless transmission of the preceding packet, so it does not contribute to the total time. On the receiving serial port, the transmission time of the first packet is masked by the wireless transmission of the following packet, so it also does not contribute to the total time. In this way both forward and backward masking occurs, and the additional cost of sending two packets compared to the cost of sending one is only the extra wireless transmission time, with no extra delay contributed by the serial port communication.

From examination of Figure 6 and our measurements, we find that the wireless transmission time for each packet is about 25ms + 250µs/byte. This is within measurement error of 246µ/byte calculated in Section 4.1. Thus this measurement confirms that in practice we can achieve a wireless transmission time of only about 32 Kbits/second, well short of the theoretically possible 100 Kbits/second.

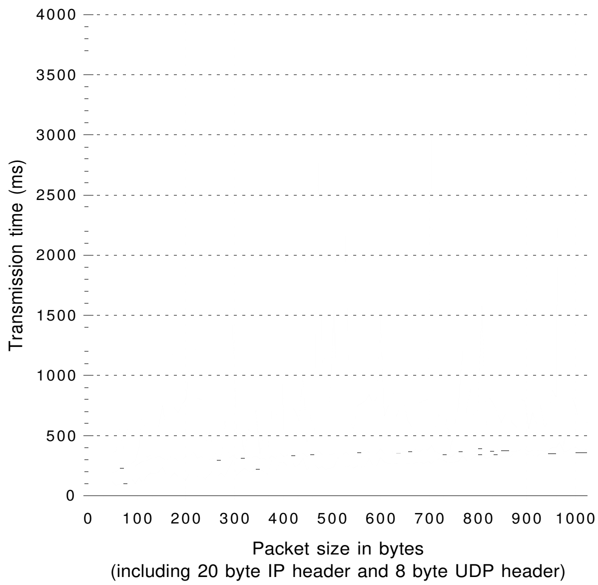


FIGURE 7. Modem emulation performance. This figure shows the minimum, average, and maximum one-way transmission times for packets of different sizes sent using SLIP over a reliable 'modem emulation' connection. At each data point we performed 64 one-packet tests. The lowest line shows the best time for the given size. The middle line shows the average time at that size, and the top line shows the worst time.

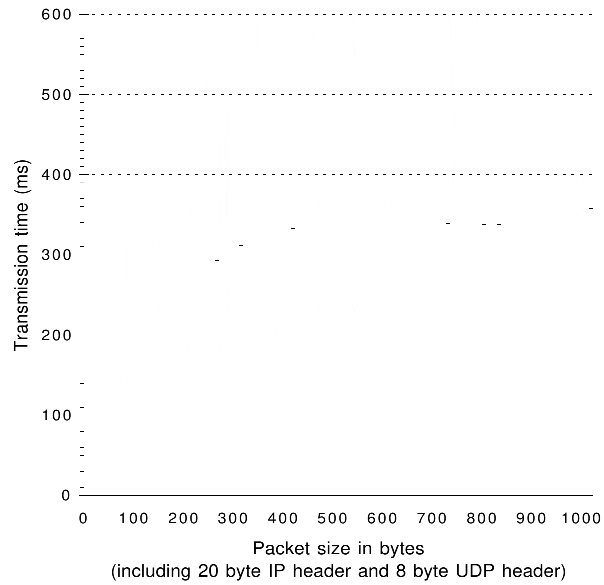


FIGURE 8. Modem emulation performance. This figure presents the same data as in Figure 7, but plotted to the same scale as Figure 2 for comparison purposes.

4.3 Modem Emulation Performance

For comparison with Starmode, we also report the time to deliver individual packets sent via the standard Linux SLIP driver, using Metricom's Hayes Modem emulation. These measurements show that the modem emulation mode achieves somewhat better wireless transmission throughput than Starmode (40 Kbits/second), but at the cost of much higher variance in packet latencies.

Figure 7 shows the minimum, average and maximum one-way transmission delay measured for packets sent via SLIP, using the Metricom radios to emulate Hayes modems. The most obvious observation from the graph is that the worst case performance is very bad, with packet delays of up to almost 4 seconds. (For easier comparison with Starmode performance, Figure 8 shows the same data plotted to the same scale as in Figure 2.)

We do not fully understand the reason for the higher variance in latency. Some of the delays are due to packet retransmissions, but since we observe a packet loss of well under 1%, we do not believe this accounts for all of the variance. However, even one lost packet can cause a significant delay. In modem emulation mode, the radios provide a reliable byte-stream connection and retransmit any lost packets. If a packet is lost the entire byte-stream is held up while the radio retransmits the missing portion. This higher reliability (which may not be required by all traffic) is bought at the cost of higher worst-case delay and higher variance in delay.

This variable delay may be even worse for higher-level network protocols. Transport protocols like TCP will treat the unexpected delay as indicating packet loss and will react by retransmitting, thereby building up even more queued data and making the delay even worse [3].

The second observation is that although the graphs are far more noisy, the best-case performance is better than the best-case Starmode performance for packet sizes above 500 bytes. This tells us that the wireless transmission is actually capable of exceeding 40 Kbits/second, but is currently unable to do so when the radio is in Starmode.

5. Interface Design Issues

This section considers one of the lessons in software/hardware interface design we learned from working with the Metricom radios. The lesson is that an interface designed for humans operating at human speeds, such as a modem interface, is not appropriate for a software network driver operating at computer speeds. The radios have a modem interface so that Metricom can penetrate the modem market easily, but this interface makes programmatic control of the radios in packet mode more difficult.

There are at least two difficulties that result from this interface. The first is that it is stateful, which makes error recovery more difficult. Putting the radios into Starmode requires several initialization steps. Under various error conditions, the radios fall out of Starmode and must be reinitialized. It is hard for the software to keep track of the radios' state to know when to trigger reinitialization. This is a less difficult decision for a human to make, since a human will usually notice that the system has stopped behaving properly.

The second difficulty that results from the radios' interface is interpreting error messages. At software speeds, error messages for packets sent in the past are hard to interpret without more context information. Humans do not send packets as quickly as software, and therefore have less trouble understanding the context of the error messages. The following two subsections describe these problems in more detail.

5.1 Initialization and Reinitialization

One of our goals is to make our Metricom driver software robust enough that it can recover, without manual intervention, from turning the radios off and on, replacing batteries, and various other sorts of minor catastrophes. This means we must handle error recovery transparently. Achieving this is not as straightforward as we originally assumed.

The reason this reliability is difficult to achieve is that Starmode is a state that requires several initialization steps to set up and configure for reasonable performance. When something goes wrong, the software must detect the problem and figure out how much reinitialization work to do. Unfortunately, a lack of positive acknowledgments from the radios makes it hard to detect that the radio is no longer functioning in Starmode properly.

The reason that Starmode requires several initialization steps is that the firmware of the Metricom radios is designed with emulation of a Hayes modem as the primary goal. Providing connectionless datagram services is a lower priority, and this is reflected in the programming interface. When a radio is powered on it behaves just like a Hayes modem, but with a few Metricom-specific extensions to the Hayes command set. To put a radio into a reasonable state for a network device, we must send it at least two such Hayes commands: a Starmode 'dial string' (`ATDT **starmode`), and a command to turn off character echo (`AT E0`).

The reason we must turn off character echo is that by default Starmode behaves as if there were a human controlling the radio using a terminal program and typing characters by hand. Humans need to see the characters they type echoed. However, a piece of software such as an IP packet driver does not need character echo. Echoed characters waste precious bandwidth on the serial port, and echoed characters interleaved randomly with the bytes of an incoming Starmode packet render it useless.

These initialization steps become problematic when it is necessary to reinitialize a radio due to an error condition. Such error conditions include the following situations:

- If the firmware of the radio crashes or encounters an internal error, the radio reboots itself, leaving it back in Hayes AT command mode.
- If a user puts the laptop computer to sleep and turns off the radio, then when it is turned back on it powers up in Hayes AT command mode.
- If the radio's battery runs down and needs to be replaced, then when the battery is replaced the radio powers up in Hayes AT command mode.

The software must determine when one of these error conditions has occurred and then trigger the reinitialization. Unfortunately, it is difficult to determine when the radio *is not* functioning properly, because there are no sensible positive acknowledgments that indicate when it *is* functioning properly.

Eventually we discovered a way to provoke a response out of the radio that would verify that it is working correctly. Sending a single pair of asterisks (consider the Starmode packet format shown in Figure 1) results in the following error message only if the radio is connected, powered on, and in Starmode:

```
ERR_004 Name too small or missing
```

Every ten seconds our current driver sends a pair of asterisks down the serial port. If the correct error message is not received within one second after that, the driver attempts to reinitialize the radio.

5.2 Error Messages

Error messages produced by the radio also show that the interface is designed for humans and not software. The radios give error messages such as `ERR_003 Can't resolve name` and `ERR_008 Bad character in name`. When a human types commands on a keyboard, messages like this may make sense. They mean, "What you just typed was a mistake." Unfortunately, when a piece of software sends packets very rapidly, it may have sent many other packets to other destinations before it receives the error message. The error indication is asynchronous due to the software's speed. Because the software does not know the context of the error message, it cannot know which radio name could not be resolved, or which radio name had a bad character in it.

Humans handle context more easily than software, since humans usually remember what they were doing just a moment ago. Making software store context information to decode occasional error messages burdens the common case code with unnecessary overhead. In many situations, such as the case of sending packets rapidly to several different destinations, it may be difficult or even impossible to determine accurately which packet the error message refers to unless the error message indicates this explicitly. Error messages for this sort of device should give enough information that they make sense in isolation.

6. Conclusions

Wireless communications have recently garnered a lot of attention, and wide-area wireless services promise to make ubiquitous network connectivity possible in the near future. Unfortunately, the performance of the wide-area wireless service we have experimented with is still such that it is hard to make its characteristics transparent to higher-level software. The throughput we measure in practice, when running IP over the radios' datagram service is only a third of the possible 100 Kbits/second air transmission speed. The per-packet latencies are also very high – at least 60ms per packet. Fortunately, at least some of this overhead is due to unnecessary software overhead in the radio devices and can therefore be improved.

Our experience with the radios also indicates that using packet-switched networking rather than reliable virtual circuits gives us two benefits. It reduces the need for the 'modem-pool' approach to providing network service, and it gives significantly less variability in packet transmission times, at the expense of not guaranteeing delivery. Although this requires that higher-layer protocols handle any packet retransmissions, this is not a significant burden for Internet protocols, since they already assume this task when necessary.

We have also experienced first hand the difficulties of writing software for an interface designed as a human/computer interface rather than a software/computer interface. We hope that future wireless devices and future interfaces to current devices will take these experiences into account.

7. Acknowledgments

We thank Metricom very much for the generous loan of their radios and for their terrific responsiveness to questions, complaints and suggestions. We also thank Bart Miller, Jonathan Stone and Xinhua Zhao for their suggestions and help with testing the software. In addition, we thank Jonathan Stone and Craig Partridge for their comments on this paper. This work was supported in part by Stanford's Telecom Center and NSF contract CCR-9501799.

8. References

[1] Mary Baker, “Changing Communication Environments in MosquitoNet.” *Proceedings of the IEEE Workshop on Mobile and Computing Systems and Applications,* December 1994.

[2] Various Authors, Linux Documentation Project, `http://sunsite.unc.edu/mdw/welcome.html`, 1995.

[3] R. Caceres and L. Iftode, “The Effects of Mobility on Reliable Transport Protocols.” *Proceedings of the 14th International Conference on Distributed Computing Systems*, June 1994.

[4] R. Droms, Dynamic Host Configuration Protocol. RFC 1541, October 1993.

[5] The Linux Journal, P.O. Box 85867, Seattle, Washington, 98145. Editor Phil Hughes, publisher Robert F. Young, 1 (1), March 1994.

[6] C. Perkins, “IP Mobility Support.” *Internet Engineering Task Force, Internet Draft,* July 8, 1995.

[7] M. Pettus, “Unlicensed Radio Using Spread Spectrum: A Technical Overview.” Available from Metricom, Inc., Sept. 27, 1993.

[8] J. Postel, Internet Protocol, RFC 791, September 1981.

[9] Matt Welsh, *The Linux Bible*, Yggdrasil Computing Incorporated, 1994.