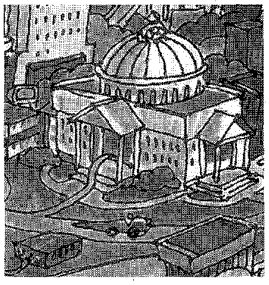
A WIRELESS NETWORK IN MOSQUITONET

Stuart Cheshire

Mary Baker

Stanford University



An emerging technology for wireless connectivity provides one of the best combinations of bandwidth, coverage area, and cost for a wireless data network. It also teaches important lessons about software-hardware interface design.

As the trend toward smaller, lighter, and more powerful computers continues, researchers on the MosquitoNet project[1] are exploring how to keep these computers permanently connected to the Internet. This connectivity will be both wired and wireless, depending on the physical location and 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 access via their portable computers to the same information and services they enjoy at 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 are to provide seemingly continuous connectivity for mobile hosts, and to support system- and application-level adaptation to dynamically changing network characteristics. We are investigating changes and extensions to standard networking application program interfaces (APIs) that will allow well-written applications to deal gracefully with widely varying network characteristics. When switching between available services, network characteristics can change significantly, even while an application is running.

For our work supporting adaptive software, we are developing a mobile-computing test bed[2] consisting of two components. First, a Mobile Internet protocol (IP) implementation based on the Internet Engineering Task Force's Mobile IP specification[3,4] allows hosts to switch between different network interfaces.[2] Second, we will provide 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 to use the best that is locally available.

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: 10-Mbps Ethernet and Metricom's Microcellular Digital Network (MCDN).[5] See the adjacent Network technologies box for an analysis of our choices. This article presents our experiences with the Metricom radios and measurements of the throughput and latency of communication using them.

The throughput of Metricom's radio devices is comparable to that of modern modems, but the latency, or round-trip delay, is higher. The maximum throughput we measured was 30 to 40 Kbps, but the minimum latency for even the smallest IP packet was at least 60 ms.

We are early users of these wireless radios' datagram mode, and our experiences confirm two sometimes forgotten principles. First, packet switching allows more efficient resource sharing 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.

Microcellular Digital Network

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

Underlying technology. Metricom's radios operate in the 902- to 928-MHz band

0772-1732/96/$5.00 © 1996 IEEE

Network technologies

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

For the wireless technology, we needed a service usable for a data network but with characteristics significantly different from Ethernet's. We chose radio devices made by Metricom Inc. of Los Gatos, California. Metricom's service is currently the only affordable one that is highly mobile, covers most of our metropolitan area, and offers performance roughly comparable to a good modem. And, unlike cellular telephone-based services, it 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 A. (The table summarizes some of the available network technologies, ranked by throughput in terms of bits per second.) We hope eventually to support as many 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 differ as dramatically from wired Ethernet. They have high bandwidths and low latencies compared to the wider-area wireless technologies, and due to their short range they are usually limited to in-building use. Since the MosquitoNet project's eventual goal is continuous network connectivity everywhere, we chose something with longer range for our first wireless technology. Of the available technologies, Metricom's radios best approximate our goal of ubiquitous connectivity.

Table A. Available network technologies.

Technology	Throughput (bits/second)	Latency (ms)	Characteristics
100-Mbps Ethernet	10^8	< 1	Wired
10-Mbps Ethernet	10^7	< 1	Wired
Wireless LAN	10^6	1 to 10	Short range
Wireless MAN	10^5	≥ 100	Slow
Wired modem	10^4	≅ 50	Slow, wired
Cellular modem	10^4	≅ 50	Slow, expensive
CDPD	10^4	≅ 500	Slow, expensive
Pager	10^3	thousands	Slow, expensive

We also considered CDPD (cellular digital packet data), but its current per-packet pricing rate of $80/Mbyte is even more expensive than the 40¢/minute connect-time cost of a cellular modem for transferring data.

Pager services such as ARDIS also provide wide-area coverage, but their current data rates and pricing make them unsuitable for a full-time Internet connection.

the FCC allocated for use by unlicensed, low-power devices. The radios use frequency hopping, jumping among 160 distinct channels in the range, and can achieve an over-the-air transmission rate of 100 Kbps.

Because each radio continually changes its receiving frequency, any radio communicating with another needs some mechanism to determine what frequency to use. A pseudorandom number generator determines the sequence of frequencies each radio uses. Every 25 ms, the radio retunes its receiver to the channel dictated by the next number in the pseudorandom sequence, thus changing channels 40 times every second.

Each radio also devotes a small percentage of time to discovering its neighboring radios. It maintains a list of neighbors and their current positions in the pseudorandom channel sequence. When a radio must 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 a radio begins packet transmission, the two radios involved remain on that channel and suspend their hopping until transmission ends. However, the pseudorandom number generator continues to run at the prescribed rate, so that the timing and sequence information other neighboring radios hold will remain correct.

Metricom's radios are packet-oriented, not circuit- or call-oriented, which allows network performance to degrade gracefully when the system overloads. Because the system selects new channels packet by packet rather than call by call, idle circuits do not tie up channels as they do in the cellular telephone network.

Due in part to the network's packet nature, Metricom charges a flat monthly fee, regardless of connection time. The company's pricing model does not discourage users from staying connected for long periods of time, because its technology supports fine-grained sharing. It makes efficient use of the available bandwidth instead of dedicating a channel per user.

Network infrastructure. Metricom's wide-area network infrastructure consists of fixed outdoor radios that are generally mounted atop 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 the installer uses a handheld global positioning satellite receiver to give each pole-top radio its precise latitude and longitude. The pole-top radios, spaced roughly half a mile apart, act as repeaters: they forward packets from one to another to reach geographically determined destinations. These radios do not use manually or automatically configured routing tables as IP routers do. Instead, a pole-top simply compares the packet's destination latitude and longitude with its own. Then, it sends the packet one hop closer to its final destination by forwarding it to the best reachable pole-top in the right direction.

Internet acronyms

API	Application program interface
ARP	Address resolution protocol
DHCP	Dynamic host configuration protocol
IP	Internet protocol
PPP	Point-to-point protocol
SLIP	Serial line Internet protocol
STRIP	Starmode radio Internet protocol
TCP	Transmission control protocol
UDP	User datagram protocol

Figure 1. Starmode packet format. Asterisks mark the beginning and end of the address. At the sending end, the address field contains the destination of the packet; at the receiving end, it contains the packet's source. A carriage return (CR) marks the end of packet. The payload begins with four characters (SRIP) that distinguish our protocol's packets from others.

Roughly 10 percent of the pole-top radios also have wired connections. If a packet must go a long way, a pole-top may 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.

Portable radios. The system provides mobile connectivity via small, battery-operated, portable radios. Each radio connects to a user's computer via an RS-232 serial port and supports rates up to 115,200 bps. The portable radios operate almost exactly like the pole-top radios, except they are not configured as repeaters.

Because the portable radios have no fixed geographic location, they require some mechanism to locate each other. Each portable radio registers with the closest pole-top radio to make its location known to other radios out of direct-communication range. Packets destined for the portable radio 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 direct-communication range, portable radios close to each other may exchange packets directly. This makes it possible for a group of Metricom-equipped laptop computers to communicate with each other in an area without Metricom pole-top service. In contrast, if you take a pair of cellular telephones (or CDPD modems) to, for example, Yosemite National Park, they are completely useless without a cellular base station.

Host interface. Metricom radios operate in two distinct modes. They can emulate Hayes modems, setting up point-to-point connections with the usual Hayes AT command set. Or, they can operate in what Metricom calls Starmode, directly sending and receiving individually addressed packets.

With the radios in the Hayes modem emulation mode, users can connect to other radios directly, 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 reliable byte-stream connection over the underlying packet-switched wireless network to emulate a modem call. Metricom provides this support because it allows users to substitute a set of radios for standard Hayes modems with little, if any, change to their software.

For our purposes, however, Starmode is more interesting, because users can individually address Starmode packets to specific destinations without any prior connection setup. In this mode, each radio behaves much more like a true network interface. Because this mode 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 nonmobile, 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, we need only connect one Metricom radio to the router. We use the radio in Starmode, allowing the router to remain in communication with any number of mobile hosts simultaneously. The capacity of the single radio to route traffic from multiple mobile hosts depends on the workload the hosts present to the network. So far in our test bed, with four active mobile hosts transferring files and reading e-mail, we have not reached the system's capacity.

In contrast, if we used the radios (as currently designed) as modems, we would need one modem per active client, just as dial-up SLIP servers do. Thus, we would need to connect our router to an entire "modem bank" of radios for our mobile hosts to "dial in" to. 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. New wireless technologies give us the opportunity to design ideal systems rather than mimic existing suboptimal solutions.

Using Starmode. The Starmode packet format is very simple, as shown in Figure 1. An asterisk precedes and follows the packet's address field. The packet's payload can contain any data. We adopted the convention of beginning the payload with a characteristic four-character code, so that we can easily distinguish our packets from unrelated Starmode traffic. 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. This means that we must use a byte-stuffing algorithm to eliminate this byte value whenever it appears inside a transmitted packet's payload and automatically reinsert the byte into the payload at the receiving end when the packet arrives.

The address field is the most curious part of a Starmode packet, because its meaning depends on whether the host is sending or receiving the packet. The address field contains a radio's name in ASCII text. (Each radio has a permanent name, usually a pair of four-digit numbers, as shown. Users may also assign additional names if they desire more descriptive identification.) From the radio sending a packet, the

address field contains the destination radio's address. On the receiving end, it contains the source radio's address. Starmode is not the actual over-the-air packet format the radios use, but the programming interface by which a host computer communicates with the radio over the serial port.

Protocol implementation

To use existing IP applications with Metricom radios, we must encapsulate IP packets using the Starmode interface. Our protocol, STRIP (Starmode radio IP), uses a straightforward encapsulation scheme similar to Ethernet's. It simply sends the IP packet as the payload of a Starmode packet addressed to the correct destination.

Since 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. (A SLIP driver does not normally include any addressing code, since it 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, though they are radio devices, Metricom radios have no broadcast ability, so a solution like the Ethernet ARP protocol is inappropriate. The radios' independent channel hopping, designed to minimize interference between simultaneous transmissions to different radios, makes it impossible for all radios to receive a single transmission simultaneously. In our current software, we must manually administer the address translation tables. However, we are now incorporating an automatic directory service into a DHCP[6] server that manages dynamic address assignment.

Performance measurements

In measuring throughput and latency for IP traffic over the Metricom radio interfaces, we were interested in how much of the 100-Kbps air transmission rate we could actually achieve. Our tests show that current radio firmware overhead limits STRIP throughput to at most 32 Kbps.

We measured 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 incurred less overhead than the first. We also found that per-packet latencies are very high—at least 60 ms in the current firmware for even the smallest packets. This includes the serial interface overhead on the sending and receiving radios. For comparison, we also measured transmission times with the radios in modem emulation mode. Modem emulation throughput is marginally better, but has much higher variance and much lower worst-case throughput.

For all of our tests, we measured one-way transmission delay, rather than the normal round-trip delay the Unix "ping" command measures. Since the radios each have only one antenna, they cannot send and receive simultaneously and are thus half-duplex devices. We measured 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.

To measure the one-way transmission delay, a daemon on the receiving side sends back a small acknowledgment over the Ethernet for each packet it receives. Because the transmission delay over the Ethernet (1 ms) is insignificant com-

Mobile IP

Mobile IP gives your computer the ability to keep a constant "home" IP address, regardless of your current point of attachment to the Internet. A constant home IP address allows you to connect to one network and then switch to another without losing your established TCP connections. Used simply, Mobile IP allows you to switch your laptop computer from one Ethernet subnet in your office to another in a meeting room. It also allows you to switch to a wireless network if you leave your office and no wired Ethernet connection is available. Mobile IP operates equally well whether the networks you switch between are wired, wireless, or one of each.

The IETF Mobile IP draft[5] is not the first attempt at solving the Mobile IP problem. Unlike most previous attempts, however, it preserves the Internet's "end-to-end" philosophy. The Internet's economy and efficiency stem from the network's bare simplicity. Users implement extra complexity at the endpoints outside the network, where it is easier and less expensive to do so. Endpoint TCP software, rather than the Internet itself, performs even such simple functions as sequencing, to ensure that packets are delivered in the right order, and check-summing, to ensure they are not corrupted.

Previous Mobile IP proposals postulated updating every router in the Internet to be aware of mobile computers and perform special routing for packets for them. Such a widespread change is logistically prohibitive. It would also burden the routers—whose main purpose is simply to deliver IP packets as quickly as possible—with an additional task. This contradicts the end-to-end design principle that originally made the Internet a success.

In contrast, the IETF Mobile IP proposal works in the current Internet with current routers. The basic idea is simple: Any mobile computer has a permanent home IP address. When it is away from home, it registers the address of its current Internet connection with a proxy on its home network called the home agent. The home agent intercepts all packets that arrive addressed to the mobile host and forwards them to the temporary IP address.

This solution is very robust—all existing Internet hosts and all existing software can communicate with a mobile host without even knowing that it is mobile—but it is not as efficient as it could be. As described, all traffic must go via the home agent. To remedy this deficiency, a "smart" correspondent host may find out the mobile host's current temporary IP address and send it packets directly. However, this optimization, and others like it, are optional. Such end-to-end optimizations are not required for correct operation and do not postulate any changes to router software.

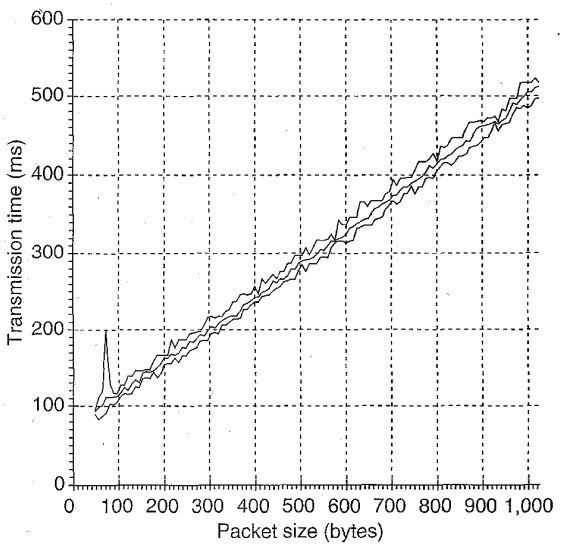


Figure 2. Minimum, average, and maximum one-way transmission times for STRIP packets of various sizes (64 bytes to 1 Kbyte, including 20-byte IP header and 8-byte UDP header). The packet size is the entire IP packet excluding the Metricom header. At each data point, we performed 64 one-packet tests. For a given size, the lowest line shows the best time, the middle line shows the average time, and the top line shows the worst time.

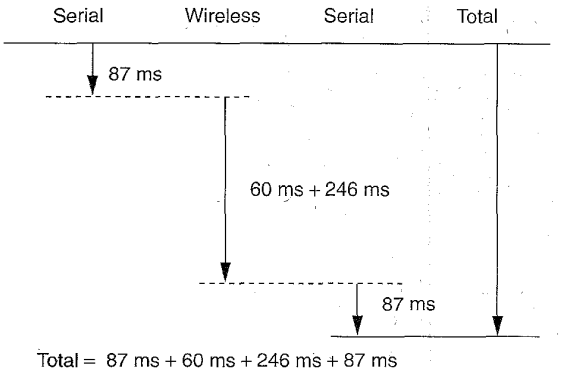


Figure 3. Components of packet latency. The total time to send a 1,000-byte packet from one radio and receive it on another includes a serial-port delay to deliver the packet from host to radio, wireless transmission time (or air-time), and a second serial-port delay to deliver the packet to the host at the receiving end. The times shown are for a 115,200-bps serial interface.

pared to the delay over the wireless interface, this yields an effective measurement technique.

We performed all of our tests 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.

IP packets over Starmode. We first measured the time it took to send individual packets of various sizes. This gave us both throughput and latency measurements, including the serial interface overhead and the data airtime. Figure 2 shows minimum, average, and maximum one-way transmission delays for IP packets from 64 bytes to 1 Kbyte. The packet size does not include the Metricom header, since it is unavoidable fixed overhead for any data sent over the radios. For each packet size, our tests sent 64 separate packets. For each group of packets, we recorded the minimum, average, and maximum delivery times.

An extrapolation of the best line intercepts the time axis at 60 ms and has a gradient of 420 µs/byte. This tells us that the time to send a packet from one host computer to another is at least 60 ms of fixed overhead plus an additional 420 µs/byte. The 420 µs/byte translates to a maximum throughput of 2,380 bytes/second or 19 Kbps, a long way short of the 100 Kbps air transmission speed.

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

Figure 3 shows the component times that make up the total transmission time for a packet sent between computers. It takes a nontrivial 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 (1 start bit + 8 data bits + 1 stop bit = 10 bits per byte) over a serial port at 115,200 bps is 87 µs. Given the 60-ms constant overhead and a 87-µs/byte serial delay for sender and receiver, the wireless transmission time must be 60 ms + 420 µs/byte − (2 × 87 µs/byte) = 60 ms + 246 µs/byte.

Even assuming that an arbitrarily long transmission would amortize the fixed overhead, 246 µs/byte translates to a maximum throughput over the air of 4,058 bytes/second, or 32 Kbps. This is still a long way short of Metricom's air transmission speed of 100 Kbps. We performed further isolated tests that indicate the overhead is not in our software but in the radio firmware supporting Starmode.

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

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 an eight-packet burst. It shows that the cost to send one 1,000-byte packet is about 500 ms, but each additional 1,000-byte packet adds only 300 ms to the total time. Illustrating this observation more clearly, Figure 5 shows the incremental cost, or additional cost, for each packet in the burst.

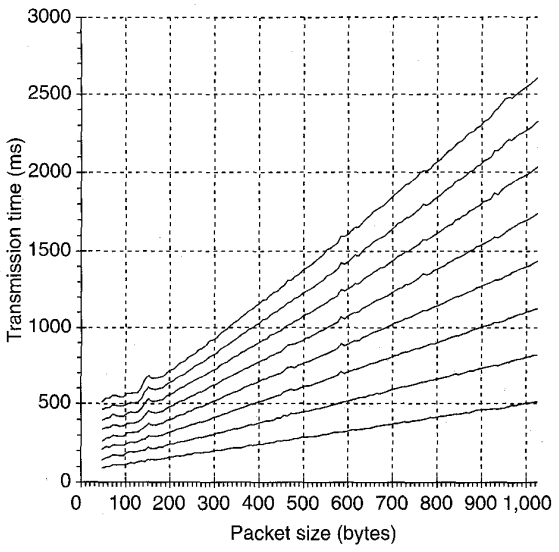


Figure 4. Packet burst transmission times. One-way transmission times for bursts of packets of sizes from 64 bytes to 1 Kbyte (including 20-byte IP header and 8-byte UDP header). The lowest line for each packet size shows the transmission time for one packet. The next line shows the transmission time for two packets, and so on. The top line shows the transmission time for the entire burst of eight packets.

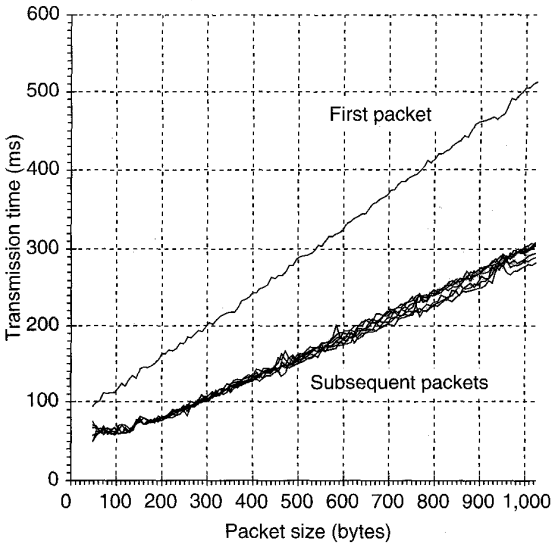


Figure 5. Incremental packet burst delays. The cost for the first packet in a burst and the incremental costs for the subsequent packets in that burst. As in Figure 4, each packet burst consists of eight packets of a given size. The top line shows the cost of sending the first packet. The lower lines show the additional cost for sending each subsequent packet.

That we see some benefit from pipelining is not surprising, since this phenomenon occurs in many other networks. For instance, TCP/IP works better than Novell Netware over long distances (links with a high delay-bandwidth product, such as satellite links) 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 additional pipelining benefits after the second packet. One might expect that it would take at least three packets to get full concurrency from a three-stage pipeline.

Figure 6 illustrates why the second packet realizes the full benefit of the pipeline. The first packet's wireless transmission masks the second packet's transmission on the outgoing serial port. Similarly, the second packet's wireless transmission masks the first packet's transmission on the receiving serial port. In this way, both forward and backward masking occur. The additional cost of sending two packets compared to the cost of sending one is only the extra wireless transmission time; serial port communication contributes no extra delay.

Examining Figure 6 and our measurements, we find that the wireless transmission time for packets 200 bytes or larger is about 25 ms + 250 μs/byte. This is within measurement error of the 246 μs/byte we calculated earlier. Thus, this measurement confirms that in practice we can achieve a wireless transmission time of only about 32 Kbps, well short of the theoretically possible 100 Kbps. Note that below 200 bytes, the curve flattens out and does not drop below 60 ms. No

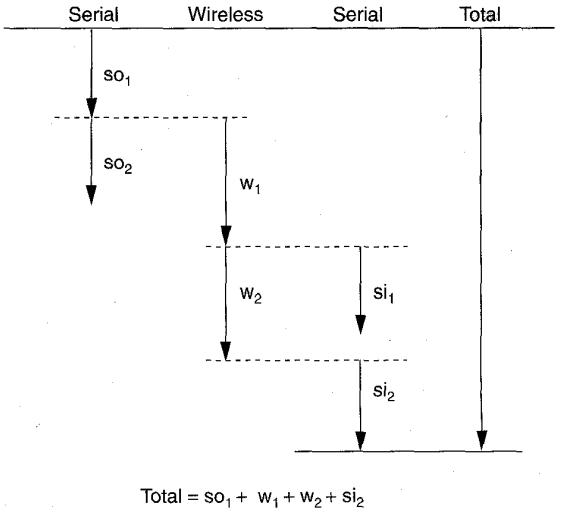


Figure 6. Pipeline timing diagram for a pair of pipelined packets. The wireless transmission time of the first packet (w_1) hides the outgoing serial delay for the second packet (so_2). Likewise, the wireless transmission time of the second packet (w_2) hides the incoming serial delay of the first packet (si_1). Hence, total transmission time includes only two serial delays for the two packets, rather than four.

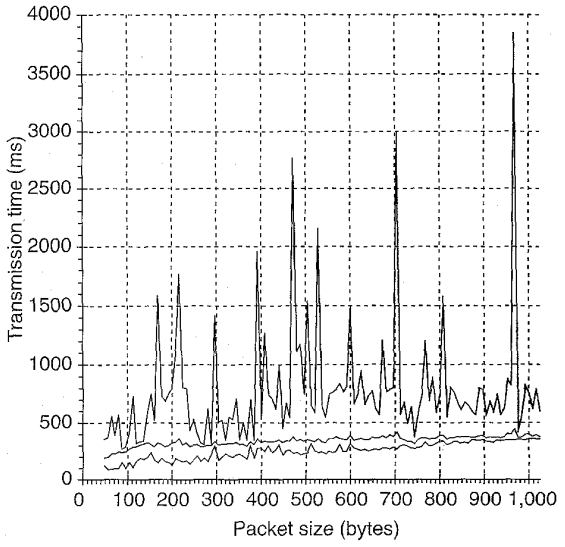


Figure 7. Modem emulation performance: minimum, average, and maximum one-way transmission times for packets of different sizes sent using SLIP over a reliable modem emulation connection. (Packet sizes include 20-byte IP header and 8-byte UDP header.) At each data point, we performed 64 one-packet tests. 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.

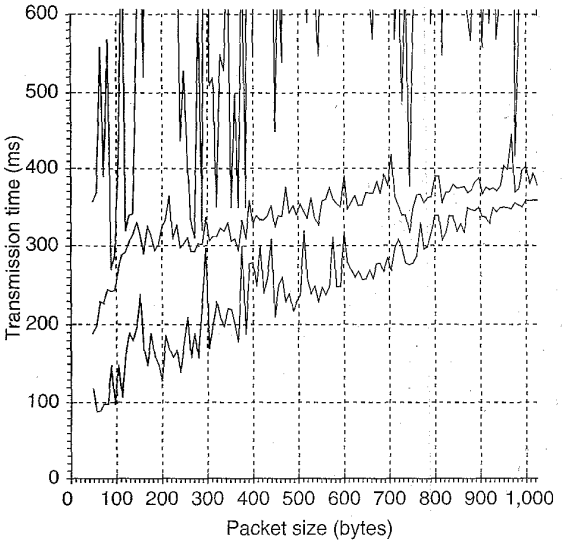


Figure 8. Modem emulation performance. For comparison, this figure presents the data from Figure 7 plotted to the scale of Figure 2.

Stop the presses!

Metricom has just announced that, as a result of the work presented in this article, it has identified key performance bugs in the Starmode software. Version 2.02 of the radio firmware, due shortly, will fix these bugs and, according to Metricom, more than double the Starmode throughput.

Initial testing by the MosquitoNet team appears to confirm a significant performance improvement. Results of detailed testing using the tools described in this article will be published, as they become available, on the MosquitoNet Web page, http://mosquitonet.stanford.edu.

packet, however small, can be sent in less than 60 ms.

Modem emulation performance. For comparison with Starmode, we also measured the time to deliver individual packets sent via the standard Linux SLIP driver with Metricom's Hayes modem emulation. These measurements show that modem emulation mode achieves somewhat better wireless transmission throughput than Starmode (40 Kbps), 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, with Metricom radios emulating Hayes modems. Our most obvious observation from the graph is that 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 data plotted to the scale of Figure 2.)

We do not yet fully understand the reason for the higher variance in latency. Although some of the delays are due to packet retransmissions, packet loss is well under one percent, so 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. Modem emulation achieves this higher reliability (which may not be required by all traffic) 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 interpret the unexpected delay as packet loss and react by retransmitting, building up even more queued data and exacerbating the delay.[7]

Our second observation is that although the graphs are far more noisy, for packet sizes above 500 bytes, the best-case modem emulation performance is better than the best-case Starmode performance. This tells us that the radios can actually exceed 40-Kbps wireless transmission, but are currently unable to do so in Starmode.

Interface design issues

Working with the Metricom radios taught us an important lesson in software-hardware interface design: 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

Staying in Starmode

Starmode, the packet-oriented mode for Metricom's wireless radio system, requires several initialization steps because the company designed the radio firmware with Hayes modem emulation as the primary goal. Providing connectionless datagram services is a lower priority, and this is reflected in the programming interface. When a radio powers on it behaves just like a Hayes modem, 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).

By default, Starmode behaves as if there were a human controlling the radio with 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 the software must reinitialize a radio due to an error condition. Such error conditions include the following situations:

- If the radio firmware crashes or encounters an internal error, the radio reboots itself, returning to Hayes AT command mode.
- If a user puts the laptop computer to sleep and turns off the radio, it powers back up in Hayes AT command mode.
- After the user replaces a radio's battery, the radio powers up in Hayes AT command mode.

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

Our solution is to provoke a response from the radio to 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_04 Name too small or missing."

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

MosquitoNet hardware and operating system

The current hardware platform for our testbed consists of industry-standard x86 personal computers running Linux, a free, Unix-like operating system. Our desktop machines, which serve as IP routers connecting the wired and wireless networks, are PCI-bus personal computers with 90-MHz Pentium processors. Our mobile computers, which also run Linux, are Gateway 2000 Handbook computers that weigh under three pounds and have 40-MHz 486 processors and 20 Mbytes of RAM.

Suggested readings

Linux Documentation Project, http://sunsite.unc.edu/mdw/welcome.html, 1995.

The Linux Journal, Phil Hughes, ed., Vol. 1, No. 1, Mar. 1994.

M. Welsh, *The Linux Bible*, Yggdrasil Computing Incorporated, San Jose, Calif., 1994.

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.

At least two difficulties result from this interface. The first is that it is stateful, which means that the interface builds up some state information required for continuing correct communication, making error recovery more difficult. Putting the radios into Starmode requires several initialization steps. Under various error conditions, a radio can fall out of Starmode and must be reinitialized. It is hard for the software to keep track of the radio's state and sense when to trigger reinitialization. This would be a less difficult decision for a human to make, since a human usually notices when the system has stopped behaving properly.

The second difficulty resulting from the radio interface is error message interpretation. 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.

Initialization and reinitialization. One of our goals was to make our Metricom driver software robust enough to recover, without manual intervention, from radios being turned off and on, battery replacement, and various other minor catastrophes. This means we must handle error recovery transparently, which is not as straightforward as we originally assumed. See the Staying in Starmode box for the details of our solution. In contrast, replacing a battery or turning a radio off and on in modem emulation mode will hang up the connection.

Error messages. The radios also produce error messages demonstrating that the interface was designed for humans and not software. The radios give error messages such as "ERR_03 Can't resolve name" and "ERR_08 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 software's speed renders the error indication asynchronous. Because the software does not know the context of the error message, it cannot know which radio name the receiver could not resolve, or which radio name contained a bad character unless the error message indicates this explicitly.

One solution, making software store context information to decode occasional error messages, burdens the common error-free case with unnecessary overhead. A better solution is for error messages to include enough information to make sense in isolation.

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 investigated is still such that it is hard to make its characteristics transparent to higher level software. The throughput we measured in practice, running IP over the radio datagram service, is only a third of the possible 100-Kbps air transmission speed.

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 network service, and it gives significantly less variability in packet transmission times, at the expense of not guaranteeing delivery. Although higher layer protocols must 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 firsthand 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.

Acknowledgments

We thank Metricom for the generous loan of their radios, for their help, and for their responsiveness to questions, complaints, and suggestions. We also thank Bart Miller, Jonathan Stone, and Xinhua Zhao for their suggestions and help testing the software. Finally, we thank Jonathan Stone, Craig Partridge, and Van Jacobson for their comments on this article. Stanford's Telecom Center and NSF contract CCR-9501799 partially supported this work.

References

1. M. Baker, "Changing Communication Environments in MosquitoNet," *Proc. Workshop on Mobile Computing Systems and Applications,* IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 64-68.
2. M. Baker et al., "Supporting Mobility in MosquitoNet," to be published in *Proc. 1996 Usenix Conf.,* Usenix Assn., Berkeley, Calif., 1996.
3. C. Perkins, "IP Mobility Support," Internet Engineering Task Force, Internet Draft, July 8, 1995.
4. J. Postel, *Internet Protocol, RFC 791,* Sept. 1981, http://www.cis.ohio-state.edu/htbin/rfc/rfc791.html.
5. M. Pettus, "Unlicensed Radio Using Spread Spectrum: A Technical Overview," available from Metricom, Inc., Los Gatos, Calif., Sept. 27, 1993.
6. R. Droms, *Dynamic Host Configuration Protocol, RFC 1541,* Oct. 1993, http://www.cis.ohio-state.edu/htbin/rfc/rfc1541.html.
7. R. Caceres and L. Iftode, "The Effects of Mobility on Reliable Transport Protocols," *Proc. 14th Int'l Conf. Distributed Computing Systems,* IEEE CS Press, 1994, pp. 12-20.

Stuart Cheshire is a PhD candidate in the Computer Science Department at Stanford University in the area of networks and operating systems. He is currently concentrating on issues of computer mobility and wireless networks. Cheshire has worked at Madge Networks, the US Army Research Institute, and Apple Computer. He is the author of Bolo, the popular multiuser, real-time Internet tank battle game.

Cheshire received his first class honors degree from Sidney Sussex College, Cambridge, and is a member of the ACM and the LPF, the League for Programming Freedom.

Mary Baker is an assistant professor in the Departments of Computer Science and Electrical Engineering at Stanford University. Her interests include operating systems, distributed systems, and software fault tolerance. She is now leading the development of the MosquitoNet mobile and wireless computing project.

Baker received a BA degree in mathematics and MS and PhD degrees in computer science from the University of California, Berkeley. She is a member of the IEEE, the ACM, and Usenix.

Direct questions concerning this article to Mary Baker, Computer Science Department, Gates Building 4A, Stanford University, Stanford, CA 94305-9040; mgbaker @ cs.stanford.edu.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 165 Medium 166 High 167