Wit: An Infrastructure for Wireless Palmtop Computing

Terri Watson
elf@cs.washington.edu
Department of Computer Science and Engineering
University of Washington
Seattle, Washington

Abstract

Wit is a system infrastructure that supports wireless connection of palmtop computers to a wired local area network (LAN). The system includes a primitive windowing system, non-preemptive threads, reliable and unreliable connections, and a Tcl interpreter[Ouster94] that allows applications to dynamically extend the palmtop system and user interface by downloading sequences of Tcl code. Together, these services provide an environment for exploring issues in mobile computing for small devices by supporting a wide range of solutions for hiding low bandwidth constraints, and permitting disconnected operation through local execution of application functions.

1 Introduction

Mobile computing is one of the hot-topics in today's increasingly computer-sophisticated society. Aggressive visionary advertising, such as AT&T's "You will" campaign, is raising user expectations for fully connected personal communicators that have CD quality audio, full-motion video, and software agents that do everything from babysitting to managing your stock investments. It's not surprising that the phrase *mobile computing* encompasses a wide variety of computing environments: from users moving from one machine to another, to disconnected laptop computing, to continuously connected handheld devices. This paper, and the Wit system, focuses on the model of a user carrying a small, handheld device. The handheld can be used outside of the wireless network in disconnected mode, but is best utilized when connected to a LAN via a wireless link. Expanding wireless coverage of metropolitan areas, combined with the increasing capabilities of personal digital assistant (PDA) computers, make this model interesting from a practical standpoint.

1.1 Motivation

Communication is the primary motivation behind wireless mobile computing, specifically, communication with people and with information sources. Cellular phones and message pagers allow telephone contact with people who are away from their desks or homes. The expanding reliance on computer communications, such as email and zephyr (a timely electronic message service)[DellaFera*88], exposes the need for similar electronic contact with mobile users. Electronic messaging also permits more privacy, and less interruption in a public setting.

In addition to communicating with other people, mobile users want access to on-line data (information retrieval). There are three reasons why users need to be connected for this. First, it is

This research was sponsored in part by a Presidential Young Investigator Award from the National Science Foundation, with matching funds from the Xerox Corporation, and by Carnegie Mellon University.

often difficult, if not impossible, to determine the data of interest ahead of time and download it to the handheld. Second, even then space limitations may prevent caching all of the data. And finally, some data changes dynamically with time (e.g. airline schedules, weather forecast, stock quotes) or with location (e.g. the closest bookstore or pharmacy) or both (e.g. the nearest librarian) and the user wants the currently relevant information.

These communication needs can be partially met by mobile users connecting to the network whenever they are near a phone line or other wired network connection. However, this solution is inadequate for many applications. Synchronous messaging is not supported much of the time, so important messages cannot be delivered until the user, who is unaware of the pending message, finally connects. Also, many wired LANs are currently ill-suited to allow an untrusted machine to connect to their network. Basically, wired connections are not available where and when needed, and the user cannot be truly mobile while connected.

In keeping with the emphasis on mobility and communication, we decided to use palmtops as our mobile units. The small size is essential for a device that is kept with the user continuously. The more restricted domain of communication and information retrieval obviated the need for a general purpose Unix box, which was fortunate as such was unavailable as a handheld product.

The above reasons motivate our choice of the wireless mobile palmtop domain. There are many other areas of mobile computing that attempt to satisfy a wide range of user requirements and applications. The next section describes related mobile computing projects that are tailored for somewhat different environments.

1.2 Related Work

The previous section motivated our choice of the wireless mobile palmtop domain. There are many other areas of mobile computing that attempt to satisfy a wide range of user requirements and applications. This section describes some related mobile computing projects that are tailored for somewhat different environments.

The ParcTab [Adams*93] system developed at Xerox is an example of a fairly specialized handheld system in a highly connected environment. The applications are targeted to tasks appropriate to the size of the output device (a three inch display) and to pen input. Most Tab applications reside wholly in the wired network, with user I/O sent wirelessly to and from Tab. [1]

The Parc MPad is a sketchpad-style wireless X terminal. Like the Tab, the MPad assumes a highly connected environment where applications are run on network workstations. The MPad project however, experimented with increased local window system support, partially to improve user perceived performance. [Kantarjiev*93]

The Infopad [Truman*94], like the MPad, is used as a wireless terminal. The Infopad however, is designed to be a multimedia terminal, including special hardware for video decompression. While using the multimedia user interface, the reliability constraints of the wireless connection can be relaxed since the video and audio data can tolerate some errors.

The Coda [Satya*90, Kistler*92] and Little Work [Huston*94] projects both provide disconnected (and in Coda's case, weakly connected) operation through modifications to the network filesystem to intelligently cache files on the mobile computer. Changes made to cached files are reintegrated with the network filesystem version when a connection to the fileserver is again avail-

[1]Newer revisions of the ParcTab have more memory and facilities to download simple data and cross-compiled programs.

System	Local Processing	Size	App Location	Connectivity
Parc Tab	display routines	small	network	continuous/slow
Infopad	multimedia I/O	medium	network	continuous/medium
Parc MPad	X server	large	network	continuous/medium
Wit	extensible	medium	network/local	continuous/slow to disconnected
Coda	Unix system	large	local	periodic/fast to disconnected
RadioMail	email	medium	local	periodic/slow

Table 1: Comparison of various mobile computing systems.

able. The mobile systems in these projects are laptops that are largely used as general purpose machines.

Mobile IP [Ioannidis*91, Teraoka*91] is designed to hide the relocation of an IP connection endpoint from applications. This allows existing applications to be unaffected by the mobility, even if a TCP/IP connection is open. Mobile IP does not try to provide continued operation in the presence of prolonged disconnection, and does not reduce bandwidth requirements.

These are just a few of the projects that exist in the mobile computing domain. Table 1 compares several of these mobile systems based on the processing done on the mobile device, size, typical location of applications, and connectivity in terms of frequency and speed.

A key distinction of the Wit system is the emphasis on supporting applications that are partitioned between the palmtop and workstation. The palmtop-local processing is general purpose and extensible, allowing applications to define sophisticated remote user interfaces.

The remainder of the paper is structured as follows. Section 2 describes the implementation of Wit system. Section 3 presents some initial applications on Wit and discusses related experiences. Section 4 outlines possible future work. Section 5 summarizes and presents some lessons learned during Wit's implementation and use.

2 The Wit System

The goal of the Wit system is to provide an environment in which to explore mobile computing. In order to allow a broad range of connectivity models, it is necessary to provide an infrastructure that supports disconnected through highly connected computing. Applications running on Wit may: execute entirely on the palmtop, be partitioned between the palmtop and a LAN workstation, or execute entirely in the wired network, using Wit only for simple terminal I/O operations.

The Wit system is divided into network-proxy and palmtop components. The palmtop system provides connections, windows, threads, and a Tcl interpreter. Each palmtop is represented in the wired network by a network-proxy process on a workstation. The proxy provides connections, a Tcl interpreter, and a socket interface. Unix applications communicate with a palmtop via the Wit proxy for that palmtop.

This section motivates and describes these system components and comments on experiences in their implementation or usage. A recurring theme in the project is to use simple solutions wherever possible. Figure 1 gives an overview of the Wit system structure.

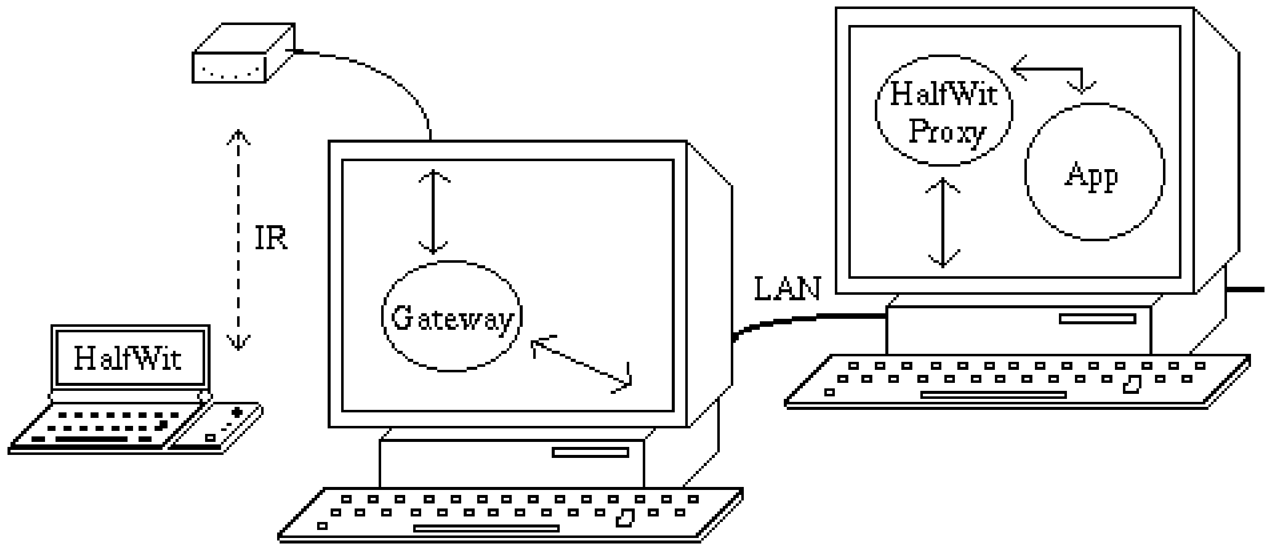

Figure 1: *Wit components. Palmtops connect to the LAN via ParcTab transceivers and gateway processes. Messages are forwarded from a gateway to the Wit proxy for the sending palmtop. The Wit proxy may handle the message, or may pass it along to an application. Applications always communicate with the palmtop via the proxy which insulates them from the complications of palmtop mobility.*

2.1 Hardware Platform

Wit is implemented on HP100lx palmtops running DOS 5.0 (mobile devices) and SUN workstations (wired network). The palmtops were a good choice for several reasons: they are commercially available, relatively inexpensive, small, and they provide a familiar programming environment. The drawbacks are that they have no standard windowing system or multi-threading support.

Wireless communication is achieved through infra-red transceivers and software developed at Xerox PARC. These transceivers communicate with their associated palmtop or workstation via a serial link at 38400 bps, and with each other using infra-red transmissions at 19200 bps. Using these transceivers allows Wit to rely on the ParcTab IRNet software, which manages location tracking and forwarding of messages from their entry point in the wired network to the proxy process associated with the mobile device. An additional benefit is the existence of a working wireless network (at Xerox PARC) in which Wit was initially developed and tested.

2.2 Communications

Wit supports both reliable and unreliable communications between a palmtop and its proxy, or between two palmtops. In the case of direct palmtop-to-palmtop communication, the connection to the LAN is not required.

2.2.1 Packet Routing

The Parc IRNet software handles the routing of packets between the palmtop and its proxy in the LAN. Uplink packets (from mobile to wired network) are sent from the wit palmtop system to the palmtop transceiver (serial), to a workstation transceiver (infra-red), to a gateway process on the receiving workstation (serial), and finally to the proxy (Sun RPC/socket). Proxy to palmtop packets follow the same route in reverse. (See figure 1.)

The link level packet header (see figure 2) contains a destination transceiver address and a sender address. The destination address indicates which transceiver will generate a link-level acknowledgement to the sender. The sender address is semantically overloaded in uplink packets – it serves as the source **and** wired network destination for the packet. This encoding is more efficient for the ParcTab system (for which the IRNet was designed) since Tabs can only communicate with their proxies. When a gateway receives a packet it uses the source address to locate the matching proxy, then forwards the packet to that proxy using Sun RPC. The IRNet code at the proxy remembers the network address(es) of the gateway(s) that have most recently sent packets from the palmtop, and uses these to route return packets. As a palmtop moves from one wireless cell to another, its uplink packets arrive at the proxy from a new gateway. When the proxy is confident that a cell switch, or "handoff", will occur, it sends a *set transceiver* message to the palmtop to update its destination transceiver address, and cell handoff is complete.

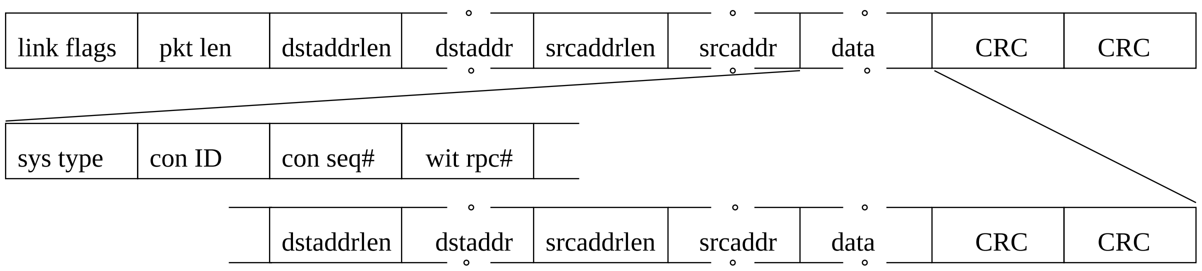

Figure 2: *Packet header fields for link and connection level encapsulation.*

2.2.2 Connections

Since the IRNet's addressing scheme only allows one fixed destination (the proxy) for uplink packets, an additional level of addressing was required to support direct palmtop-to-palmtop communication in Wit. This communication is desirable for two reasons. First, this allows palmtops to communicate with each other in the absence of the wired LAN. This is useful for exchanging information with other Wit users when away from the LAN, and for message-passing in conference rooms or locations without a transceiver.[2] Second, direct communication between palmtops significantly decreases the latency of those communications by bypassing the routing to and from the respective proxies. The additional fields in Wit's packet encapsulation are shown in figure 2.

Connections manage the state associated with each different communications channel. This state includes destination address, protocol (un/reliable), retry count and timer, sequence numbers,

[2]One of the more useful applications for the author was the ability to debug new palmtop-to-palmtop applications during a plane flight.

sliding window buffers, received packet queues, error statistics and an optional user handler for processing received packets. Although much of the code is present to support a sliding window protocol, the current implementation is restricted to a window size of one, a simple send and acknowledge protocol.

Multiple connections can be (and usually are) opened to the same destination address. These connections are paired with handlers appropriate to the application using the connection. The default handlers implement several fundemental asynchronous RPC services that are described in section 2.5.

2.2.3 Palmtop Serial I/O

Serial I/O between the palmtop and its transceiver is interrupt driven for input and polled for output. Upon character arrival, the interrupt handler is invoked and it places the character on a 4K circular input buffer. The most common transmission error observed during intensive serial I/O is a receiver overrun. The 16450 compatible UART (serial chip) on the HP100lx has only one holding register. Although the interrupt routine is short, at 38400 bps overruns occur and become more frequent during simultaneous keyboard activity. We are considering testing a 19200 bps version of the IR transceiver to try to alleviate the problem.

Output to the transceiver is poll driven. This is acceptable since the higher level software is packetizing outgoing data into fairly small chunks. Additionally, it is easier to handle certain events, such as a transceiver FIFO full. Since the transport level software is still holding the packet that caused the event, it can retransmit the packet after a short delay to let the FIFO drain.

The IR driver formats outgoing packets into transceiver send commands and handles status messages from the transceiver. It scans the circular input queue, reconstructing packets from the serial data based on framing and CRC codes. A more complete description of the IRNet transceiver protocol can be found in [Adams*93].

2.3 Palmtop Windowing

The window system on the palmtop virtualizes the screen to support multiple active applications and ameliorates the constraints of the small 80x25 character display. The local window system is also needed to adequately support disconnected operation. Wit provides overlapping text windows, with scrollable output to partially occluded windows. Windows cannot extend beyond the edge of the screen. Window specific key-bindings and event handlers (e.g. focus, output) are supported.

In practice, windows are mostly used to provide virtual screens. This occurs primarily because the small size of the palmtop text screen dictates that users really want a full screen window for each application. Local windowing has the additional benefits of improving the user perceived response of the system and facilitates local debugging.

2.4 Threads

Threads simplify the implementation of reliable communications protocols and general palmtop services, and support multiple simultaneous applications. Everything, with the exception of the serial interrupt handler, runs in the context of a thread. Wit forks 6 threads on palmtop startup to handle the following services: keyboard input, ir packet input, session keyboard input, session connection input, the tcl interpreter, and the W* browser. The original thread then blocks waiting for the system to be halted.

Wit provides simple, non-preemptive threads. Scheduling occurs when a thread calls **Thr_Yield()** or **Thr_Wait()**. The scheduler maintains one queue of threads, each of which is either active or waiting. Active threads are runnable, waiting threads have a function and argument pair (specified when they called **Thr_Wait()**) that is evaluated each time the thread is considered. If the function returns true, the thread becomes active. The scheduler selects the next active thread in FIFO order and runs it. Wit threads are implemented using C's setjmp()/longjmp() calls. A new stack and setjmp context is set-up during **Thr_Fork()** and then the thread is started by a call to longjmp().

Although the thread package was originally developed for the palmtop side of Wit, once the communications code was written on top of these threads, they were ported to the Unix side as well. The better debugging support on Unix and the desire to have a single set of sources for the communications protocols motivated this decision.

2.5 RPC

All packets on a Wit connection are treated as RPC requests. The system currently defines six types of RPC. Three of these provide output: text, audio, and video. The fourth, WitShell, spawns a Unix process and sets up a pty with the input and output directed to a Wit connection. The last two are the fundamental RPCs used for defining new services. One of these, WitTclEval, sends its data to the Tcl interpreter and asynchronously returns the result. The second, WitApp, invokes a user defined handler on the packet data. The WitTclEval RPC allows the user to dynamically customize the programming interface to meet new application requirements.

2.6 Tcl

The Tcl interpreter is the application programming interface for Wit. Tcl is a Tool Command Language developed at Berkeley. [Ouster94] The language has a C-like syntax and can be easily combined with C routines to extend the interpreter command set. In Wit, this is used to export window, connection, thread, and other routines as Tcl procedures. System parameters and error statistics can be read and updated by Tcl programs. Tcl interpreters exist in both the palmtop system and the network proxy. Remote evaluation, via the TclEval RPC permits functional extensions to the remote side by either the palmtop or proxy.

Tcl commands are executed as a result of one of five actions. The first of these occurs when the Wit system is started. Several configuration files are loaded by the interpreter; these set default timing and connection parameters, system address, etc. Customizable startup files supply common extensions to the system.

A second source of Tcl commands is the interactive interpreter window. Commands are typed into the window, evaluated, and the results are displayed in the window. (On the Unix side, the interpreter uses the proxy process's default I/O terminal (if it exists) for this.) There is only one thread servicing the interactive interpreter, so a Tcl script that loops will prevent further interactive commands, until it completes.

TclEval RPC packets are the third entry point to the interpreter. These packets can be sent from/to either the proxy or another palmtop. This support gives tremendous leverage when developing new applications as the system can be modified while active. A Tcl application can push functionality across to the remote side by defining and executing functions in the remote Tcl interpreter.

Unix applications invoke the proxy's interpreter in one of two ways. The first interface allows them to send Tcl commands to the proxy interpreter on a socket (section refsec:sockets). The second interface is only available through Tk, a Tcl based X-windows interface. Tk applications can use Tk's **send** command to send a Tcl command to a named interpreter, in this case, the proxy. This last method of invoking the interpreter can also be reversed, to send a command *from* the proxy to a Tk application. Wit's video application (section 3.1.4 uses this method to control a Unix Tk application. Figure 3 illustrates the various communications paths and general software structure.

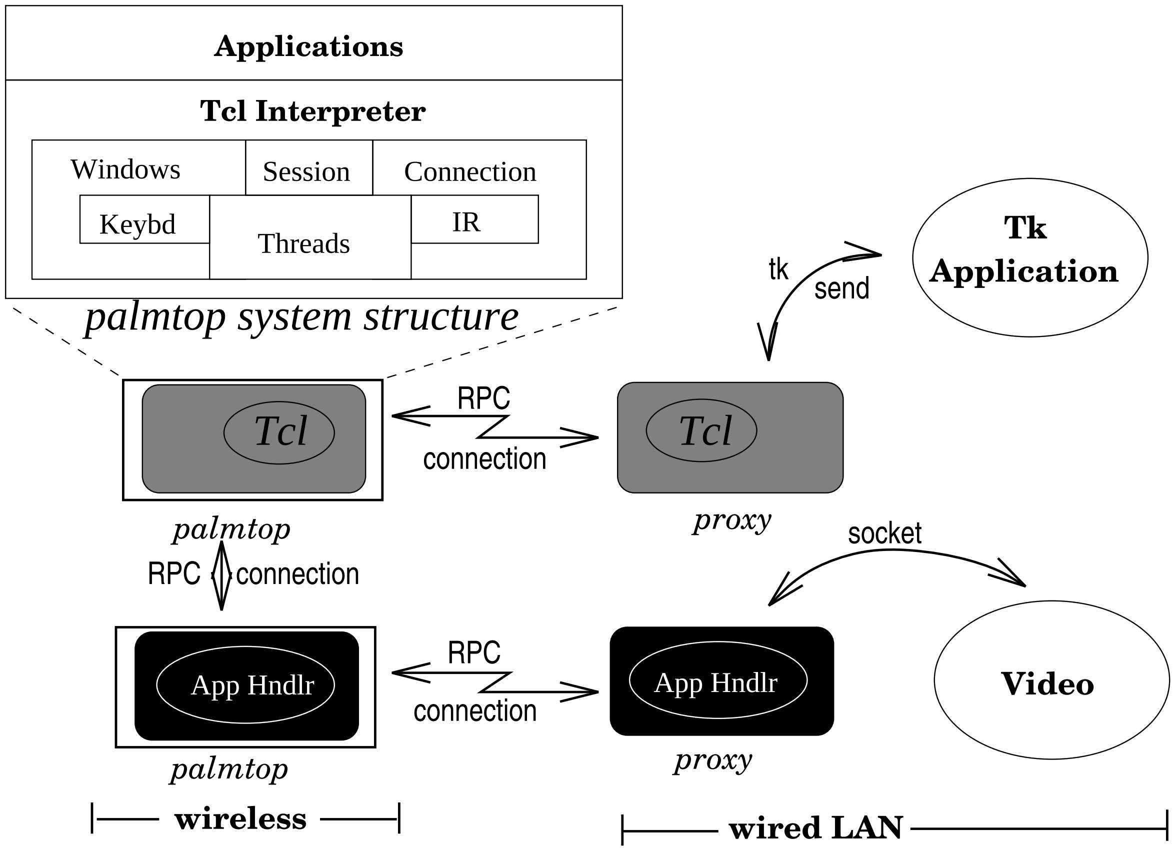

Figure 3: *Wit software structure and interfaces.*

The Tcl interpreter was a huge win in development, testing, and debugging. Interactive prototyping was useful for several applications. The access to system state variables and parameters allowed the quick construction and modification of benchmarks to collect wireless performance numbers.

Although Tcl is available on both the workstation and palmtop sides, the windowing interfaces (Wit windows and Tk) differ. The ParcTab group has started using Tcl for application development, and created a mapping from Tk windowing commands to Tab windowing primitives.[Petersen94]

Experiences with that system suggest that this would be a better approach for Wit in the future as well.

2.7 Sockets

Sockets are one of the ways in which applications communicate with the proxy and indirectly to the palmtop. They provide a means for streaming data to the palmtop. The palmtop's proxy listens on a "well-known" TCP port for new connections. The default handler for a new connection sends received data to the proxy's Tcl interpreter. The application can use Tcl to install a new handler for the socket.

A socket can be associated with a connection. This association can be used by special handlers to send socket data, possibly after some processing, over a connection, and similarly, write received connection data back to the socket.

2.8 Sessions

Windows can be associated with connections. A window/connection pair is called a session, and there are a number of system supported handlers for sessions. Default session key handlers take input from the window – either character or line-oriented, packetize it, and send it on the associated connection. Received connection data is processed by the session connection handler and sent to the window as output. Applications can specify alternate handlers for sessions.

The Wit terminal emulator (section 3.1.1) uses sessions to associate a window with the connection to the proxy, which in turn is associated with a pty for the shell.

3 Applications

This section describes some initial applications implemented on Wit, and discusses some ideas for structuring mobile applications based on our experiences with these applications.

3.1 Initial Applications

This section presents four of Wit's applications: witty - a terminal emulator, chat - similar to Unix talk, W* - a WWW client browser, and wv - a wireless video viewer.

3.1.1 Witty

The first real application was the Wit terminal emulator, witty. The terminal emulator's main benefit was increased mobile functionality. Most Unix tools have a terminal interface, so users could now read email, netnews, edit files, "talk" to other users, browse the World Wide Web, and a host of other activities. Unfortunately, witty was less than spectacular in terms of performance. The character-at-a-time nature of terminal I/O leaves few opportunities for latency hiding without some semantic compromises.

Witty provides a window option to toggle local line-oriented input. Although this disables special shell editing key-bindings, it was particularly useful in some situations (eg, meetings) where contention for IR bandwidth was extremely high.

3.1.2 Chat

Chat is a very simple application that opens a local window on the initiating palmtop and a remote window on the receiving palmtop and directs the input of each widow to the output of the other. It resembles Unix talk(1), but to keep the example simple, the output is not divided into remote and local windows. Figure 4 shows the chat Tcl procedure.

```
proc chat {addr} {
  global Pkt_WitTclEval Wit_SelfAddr

  set c [conopen $addr]
  set r [coninfo $c]
  keyin [session $c [twin 0 0 80 25 $addr]] Half
  conraw $c $Pkt_WitTclEval "keyin \[session $r
                            \[twin 0 0 80 25 $Wit_SelfAddr\] \] Half\n"
}
```

Figure 4: *Chat. The session support for redirecting window I/O to a connection greatly simplifies this application.*

Chat was the first application that used the ability to directly connect to another palmtop without using the wired network. Although a simple application, it was one of the more entertaining to use during meetings. In fact, Sega has a new product that sends messages wirelessly between two small devices in the same room. The television advertisement shows children in what appears to be a classroom setting, giggling over the messages they send back and forth – not that far off the author's experience, actually.

3.1.3 W*

W* is Wit's Wireless World Wide Web client browser. Information retrieval is one of the key applications for wireless handheld computers. The World Wide Web (WWW) [Berners*94, CERN94] contains an increasing diversity and number of information sources, from movie reviews and shopping malls to articles on medicine. Information is usually stored hierarchically in hypertext documents. The documents can contain hyperlinks to other documents, which can be automatically retrieved when the user selects the link.

The initial document fetch and file transfer for W* was prototyped using Tcl. Figure 5 shows a couple of sample procedures. Although this code is insufficient to support W*'s full functionality, it illustrates the simplicity of a workable protoype solution.

W* supports local caching and partial document loading. It also unobtrusively indicates to the user when operations may be slow. One example of this is using different attributes to display a hyperlink to a large non-cached document. W* is presented more completely in [Watson94].

3.1.4 wv

wv is a wireless video viewer. It uses a modified version of Xerox PARC's video tool, **nv**[Frederick94] to monitor a video source, and send video frames to the palmtop at an acceptable rate. The current

```
# Palmtop side
proc wsget { link } {
        global WS_Files Pkt_WitTclEval WS_Con

        if {[info exists WS_Files($link)]} {
# default fetch does not check if the cached copy is still valid
                wsdisplay $WS_Files($link)
        } {
                set tmp [uniqfile]
# a header is written to the file so WS_Files can be rebuilt on startup
                tofile $tmp "<wstar $link>\n"
                set WS_Files($link) $tmp
# sends a Tcl command to the proxy to fetch the document
                conraw $WS_Con $Pkt_WitTclEval
                               "httpget $link [coninfo $WS_Con] $tmp\n"
        }
}

# Proxy side
proc httpget { link con fname } {
        global Pkt_WitTclEval

        set tmpf [exec /homes/gws/elf/perl/hget $link]
        conraw $con $Pkt_WitTclEval "tofile $fname \{$tmpf\}\n"
}
```

Figure 5: *Prototype WWW document fetch.*

implementation performs no compression, but is still sufficient to observe large-time events, e.g. a person entering or leaving the room.

We are extending **wv** to include compression and hierarchical encoding of the video stream. Additionally, we are interested in the possibility of incorporating some determination of "interesting" frames, either based on frame differencing, such as in **nv**, or more sophisticated domain-specific scene recognition.[Zhang*94] These schemes seems ideally suited to supporting low-bandwidth video.

3.2 Application Design

Our experiences with Wit show that over a slow wireless link, the usability of applications is greatly improved by partitioning the application between the mobile device and the wired LAN. Most user interface functionality should be defined on the mobile side. This decreases the latency for user interactions, and reduces contention for the wireless link. In situations where several wireless users are in the same cell, contention may be the dominating factor on performance.

By designing an application whose mobile component can operate on local data, caching and prefetching may be used to hide or eliminate latency, and further reduce bandwidth requirements. The application becomes more tolerent of disconnections from the LAN, although possibly only a subset of operations are supported during the disconnection.

Design strategies for structuring wireless applications, using W* as an example, are presented in more detail in [Watson94].

4 Future Work

Application design for mobile computing is the primary focus of ongoing research on Wit. Most of today's Unix applications are resource intensive, and intolerant of wireless mobile constraints. Application writers are forced to manage the details of user interactions (e.g. window size, bitmap decorations) and usually make strong assumptions about the resources available (e.g. loading entire documents in Mosaic).

We are designing a high-level API that defines user-interface objects and special data types for mobile applications. These abstract objects are understood by the system, and are designed to adapt to the current resource constraints of the mobile device. For example, the system provides a hyper-object viewer. The hyper-object contains a hierarchy of objects, such as text, graphics, video, audio, hyperlinks, or other hyper-objects. The viewer can adapt to a low-bandwidth network with high-contention by "folding" lower levels, displaying only the top level objects. The user can select top-level hyper-objects to display their contents as well. During low-contention periods the system could hide latency by prefetching down the hierarchy. The component objects are able to adapt to resource scarcity as well. The video object, for instance, is a hierarchically encoded video stream. Depending on available bandwidth, the user might see a lower resolution picture, or only certain "important" video frames.

There are some basic extensions to be made to the existing Wit system, particularly if the current hardware platform is retained. The communications protocols should be updated to support sliding windows and timeout/retry values should be re-tuned. The association of connections to other I/O sources (windows, sockets, ptys) should be reworked into a single abstraction for ease of management. The Wit windowing system should support a subset of Tcl/Tk's functionality to provide a more unified interface to the palmtop and workstation. The switch to a Tk-like interface will provide a good starting point for some of the mobile API objects, and hopefully allow existing Tcl/Tk applications to be easily adapted to use the new API.

5 Conclusions

Wit provides an environment in which to explore mobile computing. Our experience with the system and several Wit applications was extremely useful for understanding some of the practical constraints of small wireless mobile devices and for experimenting with solutions to the associated problems.

Local processing and user interfaces greatly increased the usability of the system. The Tcl interpreter facilitated application defined functionality on the palmtop and was extremely useful for quick prototyping and debugging. The most commonly used applications were those that dealt with information retrieval: reading email, netnews, and browsing WWW. The most common complaint was the unpredictable latency of some operations.

Continuing work on application design should increase the usability of the system and better identify the class of applications for which palmtop wireless is appropriate and desirable.

Acknowledgements

I wish to thank Brian Bershad for laying the foundation for Wit and for continuing advice on many significant design issues.

Thanks to the the TabTeam members and others at Xerox PARC for their numerous contributions to Wit. Conversations with Norman Adams, Lawrence Butcher, Pavel Curtis, Karin Petersen, Bill Schilit, Marvin Theimer, Doug Terry, Roy Want, Mark Weiser, and Brent Welch contributed greatly to Wit's design and implementation. Barry Kercheval was a particularly wonderful Wit guinea pig, suffering through system bugs and providing valuable feedback on usability. Ron Frederick provided nv code for the video application and many helpful discussions on compression.

I also wish to thank Maria Ebling, George Forman, Garth Gibson, Lily Mummert, Hugo Patterson, Stefan Savage, and Geoff Voelker for their input on wireless application design.

References

[Adams*93] N. Adams, R. Gold, B. Schilit, M. Tso, and R. Want. An Infrared Network for Mobile Computers. *Proceedings of the 1st Usenix Symposium on Mobile & Location-Independent Computing*, pp. 41-51, August 1994.

[Berners*94] T. Berners-Lee, R. Cailliau, A. Loutonen, H. F. Nielsen, A. Secret. The World Wide Web. *Communications of the ACM*, vol 37, no 8, pp. 76-82, August 1994.

[CERN94] World Wide Web initiative at CERN.
http://info.cern.ch/hypertext/WWW/TheProject.html

[DellaFera*88] DellaFera, C., Eichin, M., French, R., Jedlinsky, D., Kohl, J, and Sommerfeld, W. The ZEPHYR notification service. In *Proceedings of the Winter 1988 USENIX Conference.*, pp. 213–19, February 1988.

[Frederick94] R. Frederick. Experiences with real-time software video compression. *Sixth International Workshop on Packet Video*, pp. F1.1-1.4, September 1994.

[Huston*94] L. B. Huston, P. Honeyman. Disconnected Operation for AFS. *Proceedings of the First Usenix Symposium on Mobile & Location-Independent Computing*, pp. 1-10, August 1994.

[Ioannidis*91] J. Ioannidis, D. Duchamp, G. Maguire Jr. IP-Based Protocols for Mobile Internetworking. *Proceedings of Sigcom 91.*

[Kantarjiev*93] C. Kantarjiev, A. Demers, R. Frederick, R. Krivacic, M. Weiser. Experiences with X in a Wireless Environment. *Proceedings of the First Usenix Symposium on Mobile and Location-Independent Computing*, pp. 117-128, August 1993.

[Kistler*92] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda Filesystem. *ACM Transactions on Computer Systems*, 10(1), February 1992.

[Ouster94] J. K. Ousterhout, *Tcl and the Tk Toolkit.* Addison-Wesley Publishing Company, 1994.

[Petersen94] K. Petersen. Tcl/Tk for a Personal Digital Assistant. To appear in *Proceedings of the Usenix Symposium on Very High Level Languages.*

[Satya*90] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available Filesystem for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4), April 1990.

[Teraoka*91] F. Teraoka, Y. Yokote, M. Tokoro. A Network Architecture Providing Host Miration Transparency. *Proceedings of Sigcom 91.*

[Truman*94] T. Truman. Infopad: A system Design for Portable Multimedia Access. *Calgary Wireless 94 Conference.* Also *http://infopad.eecs.berkeley.edu/papers/.*

[Watson94] T. Watson. Application Design for Wireless Computing. To appear in *Proceedings of the Workshop on Mobile Computing Systems and Applications.* December 1994.

[Zhang*94] H. J. Zhang, Y. Gong, S. W. Smoliar, Y. H. Tan. Automatic Parsing of News Video. *Proceedings of the International Conference on Multimedia Computing and Systems*, pp. 45-54, May 1994.