Optimising Thin Clients for Wireless Computing via Localization of Keyboard Activity

Sivasundar Ramamurthy and Sumi Helal
Computer & Information Science & Eng. Department
University of Florida
Gainesville, FL 32611-6120, USA
helal@cise.ufl.edu

Abstract

The thin client model is based on the classic client-server model with all client activity being processed at the server. Keyboard and mouse activities at the client are sent to the server, which processes the activities and sends the refreshed display back to the client. This model can be exploited for mobile computing by having powerful fixed servers serve mobile thin client devices with poor computing resources. However high latency in the wireless connection can lead to delay in display of keyboard and mouse activities, thereby affecting the performance of the thin client. This can be remedied by handling client activities locally, which will not only mitigate latency's ill effects but also reduce the volume of exchanged messages passed. This paper focuses on localization of keyboard activities in thin clients during periods of high network latency.

Introduction

At the birth of the next millennium, "Ubiquitous Computing" will become the most prevalent trend among computer users. With the exponential growth in technology and human skills, mobile computing devices characterizing lesser cost and more productivity and utility will be developed and the number of mobile computer users will increase. As this paradigm of computing continues to grow, newer problems and limitations will be encountered and efforts would have to be taken to solve them. Mobile computing, as it rightly should, serves as the main motivation behind our research on thin clients. This paper describes our research on localization of keyboard activity for wireless thin clients as a counter measure for high network latency.

In the traditional client server model, the full client is placed in the mobile host. A thin client, in contrast, is a dummy terminal that plays no role in processing application specific events. It only handles the display for the thin client process and directs all client mouse and keyboard events to the server. The server does the necessary processing, determines the new display resulting from that event, and sends the update to the client. Upon receiving these messages, the client updates the display on the host machine. Due to the content of data exchanged, thin client communication is characterized by short messages (short requests and replies) and this allows for good performance even in a weakly connected network.

- The advantages of using thin clients range over a wide domain of computing issues.
- Thin clients can run on machines with very little RAM and a relatively slow processor. For instance, Citrix's ICA Client requires the equivalent of an Intel 286 processor and access to a minimum of 640k of RAM to operate [1].
- Machines built specifically for use with thin clients can be made extremely compact and inexpensive.
- Access to Operating System specific applications can be given to thin clients regardless of the client hardware and platform [2]
- Applications can be scaled, deployed, managed and supported from a single location (the server) [2]
- Thin clients can perform under weak connections. For instance Citrix's ICA Client can function in bandwidths as low as 20kb[1].

The main disadvantages of thin clients stem from the fact that constant network connection is required for functioning and thin clients cannot operate in the disconnected mode. Also, thin clients perform poorly in networks with extreme delays and latencies; display of animation and of user’s typing and mouse activities may not be done in a time period that keeps the user comfortable and satisfied with the system.

Thin clients can be employed for mobile computing to execute applications. The amount of RAM and processing power required to execute the application is effectively the amount required for the display. Also, these applications need not be installed locally and this saves hard disk space. With the decreasing use of the CPU, the RAM, and the hard disk, the battery power required to use the three resources also reduces. This effectively enables the mobile user to increase the life of the battery and work longer before a recharge. These benefits can also be exploited to build mobile machines specifically for use with thin clients. The resulting design can be extremely compact with the CPU size, hard disk space, and RAM just about enough to run the thin client. The amount of communication usage is also kept to a minimum with thin clients. Even though the client host needs constant connectivity, the messages sent are short and represent only of keyboard, mouse, and display update data.

The drawbacks of mobile thin clients stem from the fact that constant network connectivity is required for functioning. First, wireless communication is at its embryonic stage: wireless protocols are still under development and service providers are able to charge high amounts per packet communicated. Two, wireless communication is inherently slow and the bandwidth available in most commercial wireless Wide Area Network connections is extremely low compared to Ethernet. And given that any mouse or keyboard activity results in a message being sent to the server, running thin client applications involving high volume of user activity might be expensive in two ways. The power supply is continuously drained while sending and receiving messages and the wireless network bill of the user will shoot up!

For reasons related to cost and loss of performance in slow connections, thin clients are certainly not the perfect system for mobile computing. Assuming that the cost of wireless communication is a secondary issue, a question arises on how thin clients can adapt to perform better under high latency. No thin client system can be designed to counter latency's ill effects and the satisfaction experienced by thin client users may be low. For example, a user working on a document using MS Word on a thin client may not see the display of characters typed within a reasonable amount of time. This slows down the user and as a result increases the thin client session duration! These events may lead to the user getting uncomfortable and frustrated with the system. One remedy to this problem is localization of events that may get affected by various degrees of latency. Even simple events like typing can become tasks to localize in high latency networks. In addition, localization of activities can reduce the amount of communication between the client and the server. With the high cost of wireless connections, this may be highly desirable for mobile users!

As one might have observed by now, localization of events does go against the definition of thin clients. With localization, the thin clients get fatter, which means that more burdens will now be placed on local resources than before. The client machine that was previously acting only as a display will now have to utilize its CPU and memory to help localization. A localization process needs to be deployed in the thin client and this may not need different versions in order to localize typing on all applications. With this installation may arise problems regarding version updates and control. These factors can trigger off an argument that thin clients performing localization may not only unsuitable for mobile computing but also kill some of the benefits of true thin clients. However, this may be valid little if the benefits obtained via localization outweigh the drawbacks and the increase in user satisfaction compensates for the loss of thin client benefits. In summary, genuine thin clients are useful since they place very little burden on client machine's resource but its performance can be affected by latency, a common characteristic of most wireless networks. Mobile machines that work as stand-alone PCs rely very little on communication but require more processing and memory resources and application management. In between these two ends of the spectrum is the thin client with a lean localization system that succeeds in maintaining the thin client benefits even while restricting the burden on local resources.

Localization of Keyboard Activity

Optimizing a thin client though localization of activities should approximately balance the exhaustion of the mobile machine's energy with the increased performance experienced. One of the localizations that can be performed to enhance performance is handling a phenomenon termed Keyboard Blitz (KB Blitz). A simple definition of a KB Blitz is when the thin client user types in such a speed that display of the characters is delayed for reasons such as a slow network or a slow server. As part our work, we intend processing keyboard activities locally whenever the network speed does not enable the prompt display of keys pressed. The motivation is that the user will experience better response to her typing and, as we will see during experimentation, the burden on the communication channel will reduce considerably. As a note, future work on this optimization should widen the definition of the KB Blitz to include server overloading and other relevant factors.

The simple problem definition above makes the problem domain too general and expansive for research. Since this is one of the first attempts at localization for thin clients, the target scenario for localization needs refinement. The main goal of this project is not only to display typing activity promptly but to also keep the overheads at the minimum, keeping in focus the theme of thin clients. This goal will be used as the foundation for designing the various features of the localization system.

The following have been identified as the fundamental features of keyboard localization:

- Except for subtle differences, the display of the localized typing activity must resemble the corresponding display when handled by the thin client server
- Transition from normal mode to local mode, and vice-versa, must have minimal interference on user's activity. The user may be given cues during this period to indicate transition
- The user should be made aware localization through subtle features. This is important since there will be some degradation in terms of typing activity display and some keyboard events will be handled differently than during normal mode
- The localization system should strive to implement as much of the processing as possible at the server side. This is necessary since most mobile machines are resource poor
- All typing activity that can be handled locally shall be done so. Key events that require special processing, like function keys and control combination keys, must result in a refresh and return to normal mode. These key events must be handled by the localization system as much as possible; that is, the refreshing event should take place without entailing the user to type the keys again.
- The frequency of shifting between local and normal modes should be controlled; it must not reach a level where user productivity actually starts decreasing and the client is slower than in normal mode
- When localization terminates, all typing activity handled locally must be refreshed with the server before the user continues using the client

Given these building blocks, a quick observation of typing behavior on heterogeneous applications would give us more principles to abide by. Typing in word processing applications, or what I have termed "typing applications," is most suited towards localization. There is little editing, multiple lines of text are typed using the enter key, and there is little logic involved in processing a key press. For instance, an enter key pressed in a web browser may result in a download or a movement to an arbitrary location on the page. An enter key pressed in a typing application like MS Word in most cases would result in a move to the next line and the resulting display can be handled reasonably accurately even in local mode. *The key point is that the chances of frequently shifting between local and remote modes are less with typing applications; most keys pressed in typing applications actually get displayed and those that need logical processing are few, can easily be enumerated, and the mechanisms to handle them during localization can be designed.*

It is possible to localize keyboard activity in all applications. But this rigid enforcement of keyboard localization can actually hinder rather than aid performance. For instance, if the user types in a ten character URL in a web browser application and this typing is localized, the amount of time and resource overhead needed to start and end localization might not make localization worthwhile even during high latency. With these additional observations, the fundamentals are further refined as follows:

- Only typing applications shall be targeted for localization and even extensive typing on other applications will be ignored
- The localization system must observe typing behavior of the user and should begin localization only if the behavior matches a KB Blitz (to be defined later)
- The localization display can be multiple lines based on the display dimensions of the application localized This rule will help reducing the number of transitions between modes, which may be high when the localization is restricted to one line of typing
- Moving the caret within the localization area using the keyboard or the mouse is permitted
- If the user moves the caret to any portion of the application that is not in the localized area, a refresh should occur and the eventual display must reflect the user action. Any mouse click that is not within the localized area should also result in a refresh

This section describes the heuristic issues involved in defining the KB Blitz, In order to deem the keyboard activity in typing applications as a KB Blitz, samples of the keyboard activity will undergo two tests. One, the type of keys in the sample must provide a good indication that forthcoming typing can be effectively localized. Two, the speed at which the user types in the keys must be compared to the network latency to determine if localization is necessary for prompt display. The first rule is substantiated by the fact that certain keys cannot be locally handled, and by the principle that frequent transitions between local and normal modes must be avoided. The second rule is to determine if the typing speed requires localization at all! Samples will be put to the second test only if they pass the first one

The main trade-off in testing the typing behavior for a KB Blitz is between the amount of time spent in monitoring the typing behavior and the amount of typing activity that is not localized because of monitoring. In order to make the KB Blitz test efficient and successful, the typing sample used will be the fifteen most recent keys typed by the user. While these keys give a good representation of the user's current typing mood, the sample length is also ideal for testing. It is long enough to ensure the speed of typing and the consistency in the types of keys pressed, and it is short enough to start localization as quickly as possible. It is worth noting that no sample length can make the tests fail-safe since only the user knows when she will shift from a KB Blitz mood to some other behavior!

A typing sample represents a KB Blitz only if it consists of keys that can be processed with this simple rule: the characters they represent can be displayed at the location where the user wants them to be displayed and there is no other logical processing. For most typing applications, these characters are the letters in the alphabet, special characters, numbers, and white spaces. The sample cannot be a KB Blitz if any other key event is interspersed in it. The heuristic is that samples that pass this test give a good hint that the user will continue with the same typing pattern. Any sample that fails this test is not a KB Blitz: the user's typing will continue to be monitored and the next sample will be put through the same test. Mouse events within the typing applications may also be included in the list of events that disqualify a typing sample from being a KB Blitz. This will bring the current definition of a KB Blitz more in accordance to the principle that only suitable *typing* activity will be displayed locally.

Latency in the communication channel that is below a certain threshold may not require localization of keyboard activity. Simple tests have shown that any latency of 200 ms and below is enough for the thin client to display typing reasonably quickly. While there is a short time lag between the key press and the display, latency of 200 ms seems to be a good upper limit for the "comfort zone." The localization system will follow this rule and not attempt localization when latency is below 200 ms. However, even if the network latency is above this threshold, localization may not be required if the user's typing speed does not warrant it. For this purpose, the average round trip time for a packet from the client to the server is calculated; this value is deemed to be the response time of the server and half of it is assumed to be the approximate network latency. Since latencies vary constantly in wireless networks, this process may be repeated periodically to keep the response time and latency values updated.

As for the test itself, only samples that pass the key types test will undergo it. The time frame of the sample is compared to the product of the response time and the length of the sample. If the sample's time frame is lesser, the sample passes the test and the user's typing behavior is deemed a KB Blitz. This test is also heuristic since the response time measurement process involves uncontrollable variables that can cause flaws. For instance, the actual response time may be lesser if the party at the server that participates in the response time measurement is slow to respond to the Pings used in the process. On the contrary, it is also possible that the actual response time of the typing application is higher than the response time calculated.

With the principles and rules defined above, the localization system can now be developed as software. There remains one final problem whose solution is undefined and it involves the scheme used to evaluate the localization system. The motivation of the thesis will be unsatisfied if means are not devised to measure the viability of the localization scheme. Experiments would have to be performed and their results will be compared to show the utility of the localization under various conditions. This process requires:

- Creation of an automated process that generates various keyboard behaviors including KB Blitzes. These will be the experiment benchmarks and two basic keystroke patterns would be generated. One would contain continuous typing without any events that might end localization. The other would contain events in frequent intervals that either postpone localization or result in an end to localization. The frequency of these events can be set to reflect various levels of KB Blitzes. Also, the speed of typing simulated should reflect the response time when the client is in normal mode and be a pre-defined speed ("maximum typing speed") in local mode
- The running time of a benchmark is the amount of time taken to generate all the key events in the specified pattern. The running time of the benchmarks would be recorded for various levels of client latency and these values will provide information regarding improvements the user might experience in terms of client performance
- The amount of communication between the client and the server will be recorded during each experiment and this will show how much communication localization saves
- The amount of local resources eaten up by the localization scheme during the experiments will be recorded and this should provide information regarding the cost of localization

The data collected from experiments may be combined to produce a composite value that reflects the overall utility of the localization scheme. In point 2, it is mentioned that the benchmarks would run on various latencies. This can be achieved via a simple “Network Emulator” that can artificially alter the latency in the communication channel between the client and server machines. A detailed description of this emulator will be provided in the chapter about experiments.

Localization System for Citrix’s ICA Client

Citrix's ICA technology provides the foundation for turning any client device (thin or fat) into the ultimate thin client [1]. The ICA protocol sends only keystrokes, mouse clicks, screen updates, and audio across the communication channel. While client machines need only a processor as powerful as an Intel 286 and 640KB of RAM, the communication consumes only about 20KB of bandwidth. ICA can work with any Win16 or Win32 application and is inherently platform independent. Citrix's MetaFrame server runs on top of Microsoft's Terminal Server and provides Microsoft's Terminal Services to ICA based clients. Please refer [11] for a comprehensive list of the functionality and benefits of Citrix MetaFrame.

Citrix Virtual Channel SDK

Citrix Virtual Channels are bi-directional error-free connections used for exchange of packet data between a Citrix server and an ICA 3.0 compliant client. They can be used to enhance the functionality of ICA Clients.

Virtual Channels (VCs) use the Independent Computing Architecture (ICA) protocol. Since ICA is a presentation level protocol that runs over several different transports (like TCP/IP, IPX, etc) protocols for VCs can be developed independent of the underlying transport. ICA VCs are packet-based; if one side writes a certain amount of data, the other side receives the same data when it performs a read. System software that is part of the ICA Client and the MetaFrame server manage the ICA stream and VC packets are encapsulated in this stream. ICA also provides for error correction; this ensures that data is received in the order in which it was sent.

VCs are implemented through a client-side VC Driver (VD) and a server-side VC application. All data sent from a VD to the server is done via the WinStation Driver (WD) on the client-side. The VD is actually a Dynamic Link Library (DLL) whose functions are called by the WD. Thus, it cannot initiate data transfers but can only wait for the WD driver to poll it for data and this is done frequently. When data is polled, the WD sends it to the server, where it is queued until the corresponding VC application reads it - there is no way to alert the application upon receipt of messages. All messages sent on a VC from the server side application to the WD are demultiplexed and the data is sent to the corresponding VD. The interaction between the WD and client VDs will be discussed next.

VDs are Win32 DLLs created using the VC API. The developer uses the interface to define what happens when a function from this DLL is called. When the ICA Client starts, the client engine reads the module.ini file to determine which modules to load and how to configure the various VDs. For each VD, the WD calls *DriverOpen()* and important information is exchanged during this function call: the VD gets addresses of output buffer functions while the WD gets address of the VD's *ICADataArrival()* function. Since the client runtime is single-threaded, VD developers are directed to not to use any blocking calls in the code definition: a blocking call in a function called by the WD will delay the entire client.

Once the VC is opened, the communication through it may begin. Data sent from the server application is handled by the driver through the code definition for *ICADataArrival()*. If the client wants to send data, it has to wait for next call to *DriverPoll()*. In the code definition for this function, the VD must use helper functions to send data reserve output buffer, fill it with data, and write the buffer

The list of VDs to load when the client is started can be provided in the module.ini. This file stores settings for determining VDs to load and the name of the corresponding DLL.

For more details about the VC SDK, please refer [12]

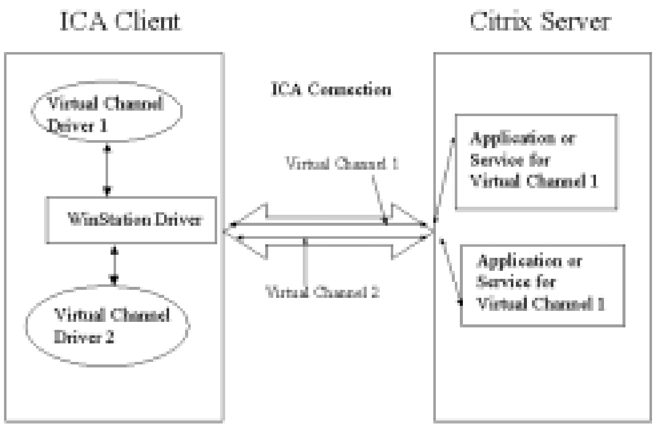
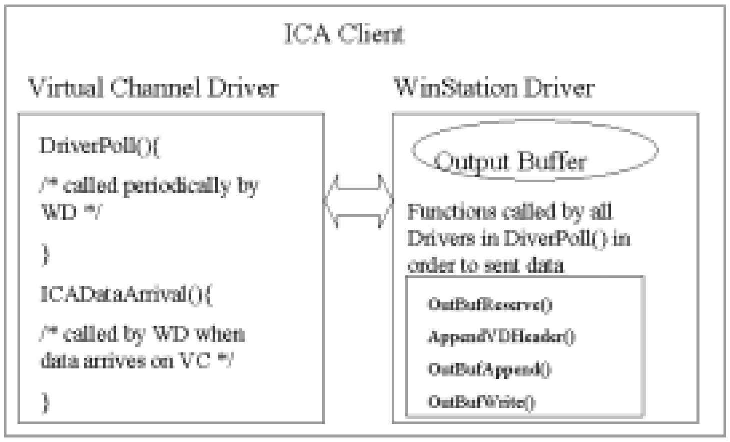


Figure 1: ICA Client at run time



```
DriverPoll(){
/* called periodically by
WD */
}
ICADataArrival(){
/* called by WD when
data arrives on VC */
}
```

Figure 2: Interaction between WinStation Driver and Virtual Drivers

Utilizing the VC SDK in designing the keyboard localization system will provide the following benefits:

- Since VCs run on top of the ICA stream used by the thin client, there is no need for creating additional communication channels between the client and the server for the localization system. Also, VCs guarantee reliable communication thus eliminating any need to monitor for possible packet losses.
- For the same reason as above, multiple VC server applications can run simultaneously on the MetaFrame server and provide the required service to their respective clients through VCs with the same name; in terms of servers, the VC name can be related to a server's "well known port."
- As VC drivers can be dynamically included to the ICA Client executable, the

localization process built using VC drivers can be easily integrated to the ICA Client as an additional functionality.

- Since VC drivers are DLLs, other applications that run completely on the client machine can be made communicate with the ICA Client by adding exported functions to the DLL. As we will see in the next chapter, one of the components of localization is an application that exploits this feature.

The *KB Pro* keyboard localization system was built to localize typing applications for Win32 Citrix ICA clients using Win16 and Win32 applications. The system is a Virtual Channel implementation and has three core components. These are the Virtual Channel Driver *VdHook*, a process called *KBWin* on the client machine, to localize typing activity, and *KBServer*, a process on the server side that serves *VdHook*. Since the localization system is for Win32 ICA Clients, all these components are Win32 modules.

The *KB Pro* system for localizing keyboard activity in the ICA Client befits implementation of Virtual Channels (VCs). VCs are of extreme use since a private communication channel between the client and server machines is crucial. Keyboard activity localized at the client machine can be refreshed with the server through a VC; also, information about the application to localize and the position of the caret in it can be sent by the server to client on this channel. In addition, the VC driver on the client side can be designed to monitoring keyboard activity in the session and detecting a keyboard blitz. Since VCs are implemented on top of the ICA stream used to run the thin client, there is no need for creating extra communication channels. And this design also results in an efficient use of the ICA channel: VC packets for different VCs can be sent together as one whole message. Finally, the localization process can be integrated into the localization system by defining exported functions in the VD that it can call.

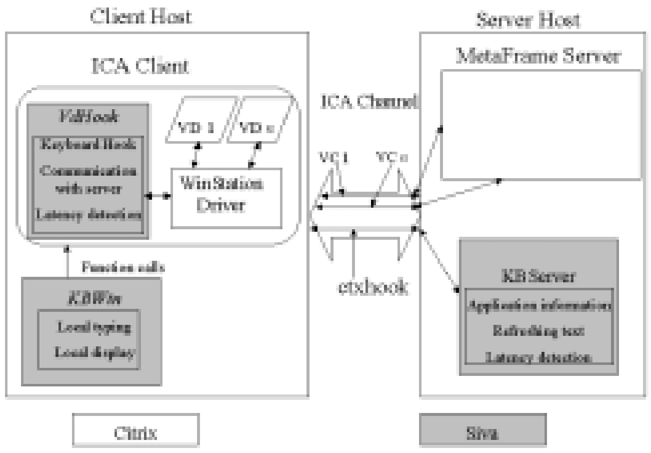


Figure 3. KB Pro system architecture

Experiments and Results

The main motivation behind the *KB Pro* system is that users of thin clients must be shielded from the harmful effects of high network latency with respect to keyboard activities. The localization system implemented enables prompt display of user's keystrokes and produces a reasonably transparent display. And even thought it maintains the thin client theme, there is unavoidable dependence on the resources of the client machine: the localization process will require hard disk space for installation and RAM and CPU power for execution. As scientists, we would like to measure the utility of such systems by experimenting under heterogeneous scenarios and producing data related to benefits and drawbacks.

Any experimentation with the KB Pro system must obviously involve typing as the main input. However, manually producing key events will not guarantee that the conditions for all experiments will remain consistent; the production of keyboard activity must be made consistent and controlled and thus automatic. For this purpose a process that will produce key events has been designed, called the *KBPro Benchmark* (we will refer to it as *Bench*). *Bench* does more just insert arbitrary key events! It does it such that the series of typing events produced emulates various typing behaviors of humans with awareness of network latency.

The inputs to the experiments must be able to create scenarios for the client to work in normal, local, and transition modes in order to measure the viability of the *KB Pro* system. These input sets, which will be referred to as benchmarks henceforth, will be produced by *Bench* using the following design scheme:

- At its bare form, *Bench* will produce a series of key events that will emulate the typing of a pre-defined sentence
- The number of times this sentence is typed can be specified as the *number of cycles*.
- In order to produce typing behaviors with fewer KB Blitzes, refreshing events can be interspersed in each cycle of the benchmark.
- A key event series is not a KB Blitz when a refreshing event is interspersed in the series. The reaction of *KB Pro* to this event during normal mode will be the same as when the user does any editing - localization is postponed.
- The event used to disqualify a series from being a KB Blitz can also be used as the refreshing event when the client is in local mode.
- The refreshing event has been designated as: Press of the alt key, press of the key representing the character 'f', and finally another press of the alt key

Another factor considered while designing *Bench* involves the transition period between local and remote modes. Remember that *KB Pro* handles user activities while shifting from normal mode to local and not during the transition from local to normal mode. Thus, *Bench* needs to be aware of such scenarios. If the client is in transition, *Bench* delays the next key event until the client gets out of transition. In order to make the benchmarks more human, the rate of key event generation was considered. Lag between key events will produce a behavior that may make the visual appearance of benchmarks more authentic. A simple observation of common typing behavior will show that users normally type a set of 4 to 5 keys with very little time lag in between the keys. However, the lag between the 5th key and 1st key of the next set is more - a distinct pause! The user normally waits for or takes notice of the display of the 5 characters before moving to the next set. *Bench* produces key events to mimic this behavior. If the client is in local mode, the time gap may between sets is appropriate to the response time of the client. During local mode, the gap is a pre-defined value that is apt for immediate display of characters typed.

Since the built in logging features of the ICA Client and the Performance Monitor are utilized for measuring communication and local resource overheads respectively, *Bench* is designed to provide the starting and ending times of a particular benchmark. This not only gives the duration of the benchmark (one of the measurements) but also provides the time bounds of the log file to focus on for obtaining overhead information.

The thin client prototype, as described in chapter 3, runs on a wireless private network with a raw bandwidth of about 2 megabits per second and a ping time measurement less than 10 milliseconds. In order to simulate the performance of the thin client in communication channels that are much slower, a network emulator has been developed. This emulator is implemented using two Virtual Channel Drivers (VDs) that communicate separately with one server side application using two VCs. The server application provides a GUI to set the network latency to particular values. The design scheme of the emulator exploits the single-threaded feature of the ICA Client. VD developers are warned against including blocking code in the DLLs since the entire client blocks and thus slows down. This feature along with that of the *DriverPoll()* function being called periodically by the WinStation Driver (WD) provides the perfect framework for artificially slowing down the client. If a VD is made to sleep for a certain time period inside the *DriverPoll()* function definition, the entire client waits for that time period and eventually slows down. This blocking code in one of the VDs delays all other client functionality, including other VDs and, most importantly, the segment that sends messages to the server and receives display refreshes. The delays that occurs from this scheme actually occur between the WD and the underlying protocol. While the messages to and from the client machine are delayed only as much as the network latency, the time it takes for the WD to write on or read from the connection is delayed; this effectively increases the latency in communication between the client and the server machines. Since this emulator is implemented using VCs, it can be utilized irrespective of the underlying protocol used by the ICA connection.

The Windows NT Workstation 4.0 operating system comes with a built-in performance monitoring tool called the Performance Monitor (we shall call it Monitor for convenience). This applications runs a GUI by which the user can set counters such as memory and processor usage to be monitored and the display will produce real-time graphs, reports, or log files as chosen by the user. When Monitor produces log files, the data from these files can be used as input for producing graphs and reports. In addition, time periods within the duration of the log file can be specified to produce data for that period. This feature is exploited during the experiments on KB Pro; since *Bench* produces the starting and ending time of a benchmark run, time periods within a log file can be clearly specified. The result is that processor and memory measurements for that time period can be extracted and added to the other measurements of that experiment. However, we should note that these values are for the entire system during the experiment and not for a particular process. Monitor does provide for specific process monitoring but with the current implementation of *KB Pro*, utilizing this feature is rendered impossible. It should be noted that Monitor does provide for monitoring other systems counters; but they are ignored them due to lack of knowledge regarding their relevance to the evaluation of *KB Pro.*

Measuring the utility of the *KB Pro* should ideally be done by comparing data collected when the client does and does not use the localization system. While the main motivation of the *KB Pro* system is to improve client performance under high latency in communication, the design of the system is based on two aspects. They are increasing user satisfaction with the thin client and keeping the burden placed on the resources of the client machine and the server at a minimum. The choice of data collected from the experiments should be based on the above two.

Since user satisfaction is subjective, choosing variables that reflect user satisfaction may produce results that are subjective also. However, when the core issue is keyboard activity and thin network latency, *time* can be designated as a measuring scale for user satisfaction (*time is money*!). Personal experiences will show that we type faster in fast telnet connections than in slow ones. One can argue that confident users who are aware of the delay may not wait or retype but this may be the case only when the amount of typing done is little. When the user is typing a document, what she types may require seeing what was typed immediately before. And for all practical purposes, the lay computer user likes to see what was typed as soon as possible. Adopting this principle, the time required to run a benchmark on a client with KB Pro can be compared to the time required to run the same on a client without KB Pro. Since *Bench* is designed in such a way that the typing behavior generated reasonably matches typing instincts of humans, the time measurements will be a good representation of the time humans would take to type in the same text. And the difference in these measurements should give us a reasonable index of increase in user satisfaction.

Apart from measurements related to time, measurements on the amount of communication between the server and the client should also provide evidence to the benefits of KB Pro. In fact, it can also be used as means to indicate increase in user satisfaction: if the number of packets exchanged is lower with *KB Pro*, then the wireless network bill will reduce and the user's satisfaction will increase! The communication done will be measured using the logging feature built into the ICA Client that will log all messages received and sent by the client. *It is assumed that the logging includes communication on all VCs since the tutorial of the Program Neighborhood application states that all packets sent and received by the client will be logged*. Since most wireless networks are packet based, the number of packets exchanged is more important than the number of bytes exchanged.

In summary, the following will be measured and compared:

- *Benchmark duration*
- *Number of messages sent*
- *Number of messages received*
- *Number of bytes sent*
- *Number of bytes received*

The remaining choice of variables will provide information with respect to the drawbacks of localization. These are burdens placed on the resources of the client and server machines such as memory and processor usage. The performance monitor built into the Windows NT Workstation operating system will be used in this measurement process. During experiments, the specific counters that will be monitored are:

- *the average percentage CPU time*
- *the average percentage committed bytes in use*

The settings for each experiment will be from the following choices:

- using *KB Pro* and without using *KB Pro* (2 choices): comparison of benchmarks with the same settings in these two groups will provide proof of KB Pro's utility.
- benchmark lengths of 2,5, or 10 cycles of 90 characters per cycle (3 choices): this will show how effective KB Pro is with different volumes of typing activity.
- network latency values of 100, 300, or 500 ms (3 choices): 100 ms was chosen to show the effect of *KB Pro*'s presence without any localization. Since even 500 ms latency proved cumbersome, higher latencies were avoided.
- refreshing event rates of 0,1, or 2 per cycle (3 choices): a refreshing rate of 0 is perfect for *KB Pro* but not too many people work this way! 1 is an ideal setting for *KB Pro* to show its "real world" utility. But 2 events per 90 characters will be an effective setting to evaluate *KB Pro* for typing behaviors unsuitable for typing.

- The amount of packets exchanged for benchmarks of the same length are very close and this is a testament to the control of experiments
- Duration of benchmarks with the same lengths and with no refreshing events are within a very close range, irrespective of latency. This typing behavior is best suited for localization and latency has little bearing on the amount of time required to complete the benchmark
- The duration ranges clearly increase when refreshing events are introduced
- The comparison between benchmarks running with latency below 100 ms shows that the duration, and packets exchanged are within a close range
- There is a consistent decrease in both duration and packets exchanged when latency and event rates increase
- The best results are for benchmarks without refreshing events
- Benchmarks with event rates of 1 and latency greater than 200 ms are realistic typing scenarios when using mobile thin clients for typing applications. The measures for these benchmarks can be marked as the best means to determining the utility of *KB Pro*
- For most cases of benchmarks with event rates of 2, the amount of bytes received is more than when *KB Pro* is not part of the client. The reason for no decrease could be due to the following: as refreshing events increase, the number of times *KBWin* starts and exits during the benchmark also increases. This means that the ICA Client window shifts back and forth from keyboard focus, which might require a repaint of the window and eventually more bytes being sent by the server. Note that there is clear decrease in number of bytes sent in the same benchmarks.

Unlike the communication and time measurements, measuring the processor and memory utilization during the experiments could not be done precisely. Preliminary observations have shown that memory and processor utilization logged by the performance monitor include spikes that sometimes reached 100% processor utilization. This can be owed to the background processes in the system, automated updates of files, etc. Few pairs of experiments, one using *KB Pro* and the other not, were identified without any spikes in either of them. Still, these give only an indication of how much resource *KB Pro* needed and, unlike the measurements on time and communication, definite trends cannot be observed in the results. The graphs shown in the next section have been chosen from these experiments and the only conclusion we can arrive is at is that localization does require *some* memory and processor resources. While this seems obvious, the only doubt in my mind was whether the communication saved by localization offset the processing power required by *KBWin.* Remember, sending and receiving messages does require some processing power! Due to the power of the processor in the client machine, the chances of observing this possibility reduced considerably.

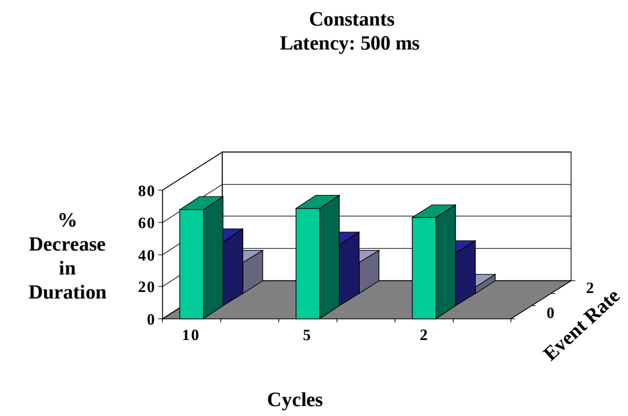


Figure 4. Graph showing results of benchmark durations

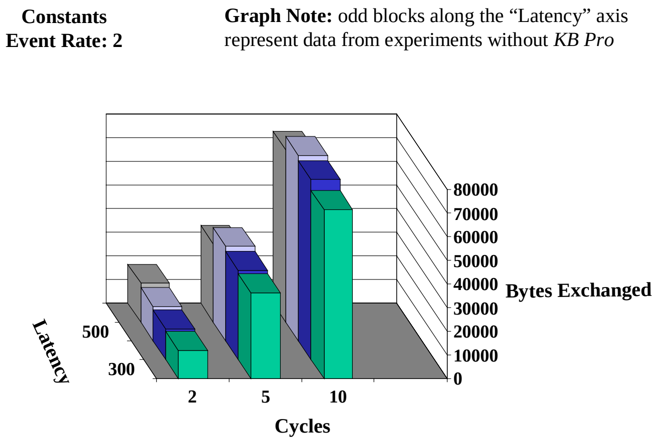


Figure 6: Graph showing effect of refreshing events on bytes exchanged

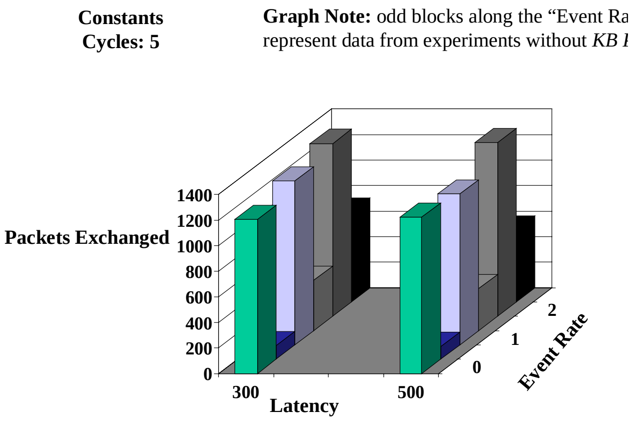


Figure 5: Graph showing effect of refreshing events in packets exchanged

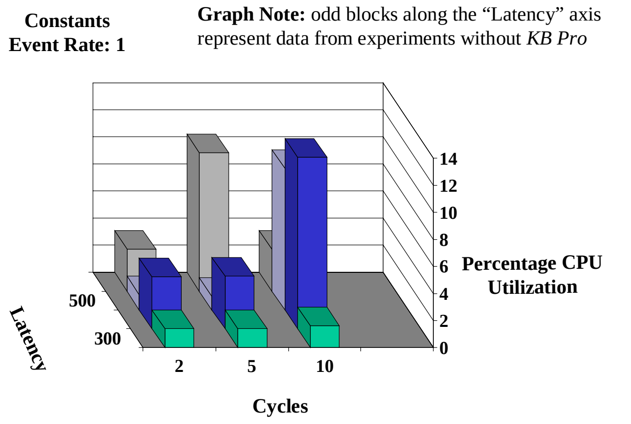


Figure 7: Graph showing average CPU utilization

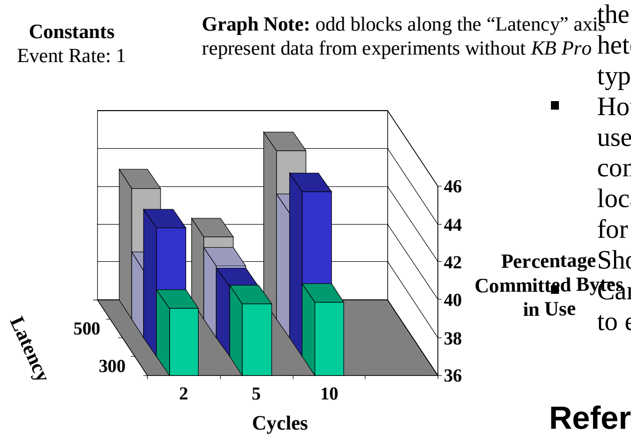


Figure 8: Graph showing average memory utilization

Conclusion

This paper has brought into light the potential of thin clients being an ideal computing tool for mobile users. It identified localization of keyboard activity during high network latency as a fertile problem for research, especially since high latency is an inherent characteristic of wireless networks. This problem definition is refined to an extent that a localization system can be designed even while retaining its core issues. The paper then described the thin client prototype, the Win32 Citrix ICA client, and its developmental tools as the platform on which the localization system could be implemented. The next section provides an overview of the implementation of the localization system for the Win32 Citrix ICA client. Then, the scheme for experimenting with the localization system to evaluate its utility is elaborated; the tools developed and utilized in the experiments are described and the variables to evaluate the system are enumerated along with the rationale behind their choices. Finally, the experimental method is described before providing the data collected from experiments and their analysis.

The following are few issues one can ponder and make cause for future work:

- Can the reliability of the scheme to monitor the keyboard activity of the user be increased, especially in improving the chances of lengthening the localization period?
- Can the capability of the localization process increase in order to reduce the scenarios that end localization? Should the versatility of the system increase with respect to heterogeneous applications or should only typing applications be targeted?
- How transparent must the system be to the user? Should the thin client system have complete control of when and how to localize? Or should an interface be provided for the user to exercise these options? Should the control be shared?
- Can more powerful experiments be designed to evaluate the viability of localization?

References

[1] www.citrix.com/products/ica.asp, May 2000

[2] Citrix's Server Based Computing White Paper www.citrix.com, May 2000

[3] www.microsoft.com, May 2000

[4] www.microsoft.com/windows2000/guide/server/solutions/terminal.asp, May 2000

[5] Citrix MetaFrame for Windows 2000 Services Fact Sheet, www.citrix.com April 2000

[6] Citrix Virtual Channel Software Development Kit Documentation