# Accelerating Telnet Performance in Wireless Networks

Barron Housel
IBM Corporation
P.O. Box 12195
Research Triangle Park
NC 27709
1-919-254-6337
bhousel@us.ibm.com

Ian Shields
IBM Corporation
P.O. Box 12195
Research Triangle Park
NC 27709
1-919-254-7200
ishields@us.ibm.com

## ABSTRACT

This paper describes the design of a system that significantly improves the performance of telnet data delivery for 3270 and 5250 emulation so that access to legacy applications via mobile units over low bandwidth wide area wireless networks is feasible. One of the key innovations of this technology is data stream caching. More generally, data stream caching can be used to reduce the transport volume for a broad class of data streams such as Lotus Notes or file transfer. The technology described herein is implemented in a system called Emulator Express (EE) [1], middleware that optimizes the operation of telnet 3270 and 5250 emulation. EE can be used with any wireless or wireline technology that implements the TCP/IP protocol. The success of the EE technology is reflected by some customers that report that the performance of running host applications when connected to their wireless networks often exceeds that of running the same applications when connected to a LAN.

## Keywords

Data reduction, caching, compression, telnet**,** wireless, mobile, sessions, emulation**.**

## 1 INTRODUCTION

A large amount of the world's data is accessible through IBM 3270 and 5250 (or compatible) display terminals which are fixed-function buffered, block-mode devices with a certain amount of built-in formatting capability. There are now many emulators of these terminals available for personal computers. The original terminals used BSC or SNA protocols to communicate with a host, but it is now common for emulators to use TCP/IP protocols and a specific telnet regime (TN3270 or TN5250) to communicate to a host computer either directly or via a separate TN3270 or TN5250 server which in turn is attached to the host using SNA protocols.

Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access, 1999, Pages 69 - 76
https://secure.acm.org/pubs/citations/proceedings/comm/313300/p69-housel/

The wide-spread use of today's emulation products testifies to the value of emulation. While new Web/Internet technology is slowly replacing the legacy systems, access to these legacy systems will be important for the foreseeable future. Indeed, new emulators, written in Java, such as IBM's Host-On-Demand , have been developed to provide seamless emulation from Web browsers. Gateway functions on various Web servers and gateway products contain functions to access host applications using SNA or telnet emulation protocols; e.g., the CICS Transaction Gateway [3] supports 3270 to access host applications.

## The Benefit

Many businesses and public services such as police have mobile field forces or officers who are in radio contact with some form of base. Typically, such folks may be dispatched to a job or location by a dispatcher who uses a 3270 or 5250 terminal connected to a central computer system. Additional information from the same or a different system may also be retrieved for particular cases, such as a criminal record check associated with a license plate check when a police officer pulls over a vehicle for a traffic infraction.

In such cases the dispatchers may become a bottleneck, particularly in crisis situations. Furthermore, a dispatcher may not relay all available information or may make errors. Finally, the voice channel itself is not always clear. For such environments, a mobile computer in the field person's vehicle is a promising way of reducing delay and increasing accuracy of information delivered to the mobile user. Ideally, such a solution should be able to access the same information as the desk-bound dispatcher could access, preferably without extensive reprogramming of the underlying information systems.

## Optimization is Required

For TN3270/TN5250 emulation to be effective over wireless networks, it is essential to significantly reduce the amount of data that is normally transferred between a host application and a 3270 or 5250 terminal emulator over such a TCP/IP connection. This reduction is particularly important in packet radio networks where a private communications protocol usually underlies the IP transport. For example, the ARDIS network uses a packet size of 240 bytes and the RAM network a packet size of 512 bytes. Both

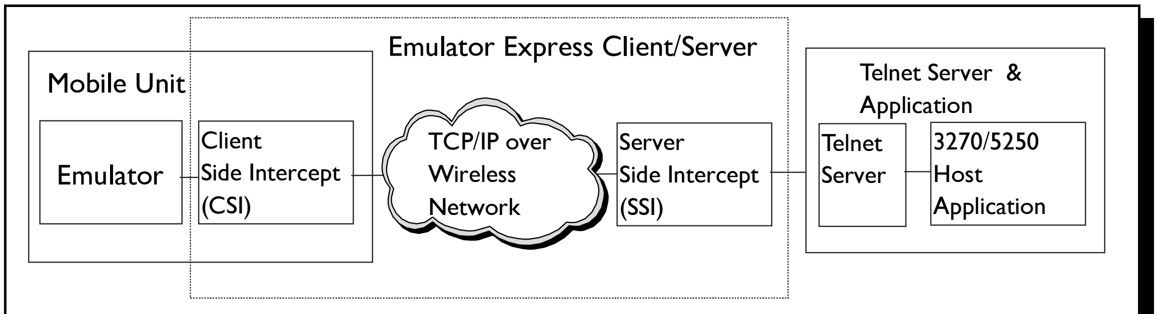
Emulator Express Client/Server
Mobile Unit
Emulator
Client Side Intercept (CSI)
TCP/IP over Wireless Network
Server Side Intercept (SSI)
Telnet Server & Application
Telnet Server
3270/5250 Host Application

**Figure 1 - Dual Proxy Model**

of these networks limit the number of packets that may be outstanding without acknowledgment. For the purpose of illustration let us assume that the time required to transmit a radio packet or acknowledgment is a constant *t*. If a message fits into a single packet it can be operated on by the receiver after time *t*. However, if two radio packets are required it may take *t* for the first to arrive, *t* for the acknowledgment to be returned and another *t* for the second packet to arrive. Adding even a few bytes to a message may thus triple the radio transfer time required before the receiver can use the message.

Traditional buffer compression and EHLLAPI[1] screen-scraping techniques have been used to achieve efficient data delivery of 3270 and 5250 data to mobile units [5][6]. One major drawback of the screen scraping approach is that not all of the 3270 or 5250 data stream is available to the EHLLAPI application. This is particularly important for 5250 applications which use advanced field characteristics such as auto-fill or auto-entry which are not available to an EHLLAPI applcation. Another major deficiency is that, for general emulation, a special emulator is needed to rebuild the display on the client device. Thus, users cannot use their emulator of choice. This approach is suited to an environment where specific application programming is done for the mobile unit to adapt the existing applications for mobile use. It is not well-suited to a general approach that attempts to reconstitute arbitrary application screens at the mobile unit as it cannot do so with fidelity because of the EHLLAPI limitations.

## Emulator Express Approach

The IBM Emulator Express product provides efficient emulation over low bandwidth network connections without the limitations described above. EE provides the significant improvement over wireless networks and also noticably improves emulator performance over modem land-line connections. More specifically, the main objectives of the IBM Emulator Express product were to:

- Optimize the delivery of 3270 and 5250 data to mobile users over TCP/IP connections using telnet (TN3270/TN5250) protocols enabling mobile clients to use 3270 or 5250 emulation over low bandwidth (e.g., wireless) connections with acceptable performance in terms of cost and response time. The use of TCP/IP based transport is consistent with the directions taken by wireless network providers: the Cellular Digital Packet Data (CDPD) protocol supports Internet Protocol (IP); the IBM eNetwork Wireless Gateway and eNetwork Wireless Mobile Client provides an IP interface over public packet radio network such as ARDIS and RAM; GSM provides TCP/IP transport over public switched networks. Several vendors manufacture 3270 or 5250 emulators that operate on a personal computer and use TN3270 or TN5250 forms of the telnet protocol over TCP/IP networks to connect to host computers.
- Operate with any vendor's emulator[2] or telnet server.
- Operate without imposing any restrictions on the data delivered to the emulator over the telnet protocol.
- Be easy to install and configure. Ease of use was considered to be very important to gain market acceptance. EE requires a minimum of additional configuration and client configurations can be updated dynamically at run-time.
- Support of multiple sessions to the same host.

Unlike screen-scraping techniques that operate on screen display buffers, the EE data reduction algorithms operate on the native 3270 or 5250 data stream directly. EE employs three optimization techniques: caching of data stream segments, compression of transmission buffers, and telnet protocol reduction. Caching and compression are aimed at significantly reducing the volume of data transmitted over the network, while telnet protocol reduction eliminates flows during session establishment to improve response time. A detail discussion of each of these techniques is the subject of the rest of the paper.

## 2 OVERVIEW OF EMULATOR EXPRESS

As shown in Figure 1, EE uses a dual-proxy configuration that consists of a *Client Side Intercept (CSI)* program located in the mobile unit and a *Server Side Intercept (SSI)* program located in the wireline network typically at or near the host system. This dual-proxy approach borrows from the IBM eNetwork

[1] The Extended High Level Language API (EHLLAPI) interface is used to access 3270 and 5250 data contained in the screen display buffers.

[2] Several vendors manufacture 3270 or 5250 emulators that operate on a personal computer and use TN3270 or TN5250 forms of the telnet protocol over TCP/IP networks to connect to host computers.

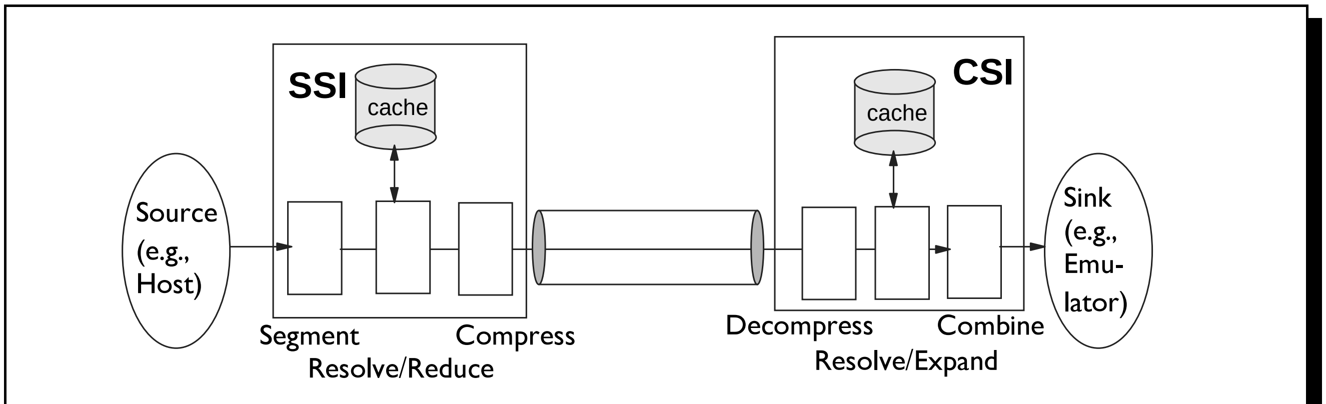

**Figure 2 - Basic data stream caching model**

WebExpress model [7][8] for optimizing Web browsing over wireless networks. The CSI-SSI pair is transparent to both the client's emulator and the telnet server (and host application). The emulator communicates with the CSI via a TCP/IP connection using the TCP/IP loopback[3] feature. The CSI and SSI communicate using a private protocol over a TCP/IP connection. The wireless link exists on this connection, and the CSI and SSI cooperate to perform data reduction. The emulator data stream seen by the emulator and the telnet server is the same as though there were a direct telnet connection between the emulator and the telnet server; i.e., the CSI reconstructs data received from the SSI into its valid telnet data. Similarly, the data stream received by the SSI from the CSI is recomposed into a valid telnet data stream that is delivered to the telnet server.

Therefore, except for minor configuration changes, the emulator and telnet server are completely unaware of the presence of the EE system. EE configuration is discussed in more detail in section 3. Although not shown in Figure 1, many mobile clients may be connected to a single SSI; likewise, a single SSI may communicate with many diffferent telnet servers and application hosts.

## Data Stream Caching, A New Technology

As shown in Figure 2, the heart of the EE system is a combination of caching and compression technology that significantly reduces the large outbound (host to client) data streams typical of 3270 and 5250 sessions. Most 3270 and 5250 applications use menus or other screen formatting techniques such that parts of the displayed image such as prompts or instructions frequently recur many times in a typical application. The EE caching technology causes these repeatable parts (segments) to be cached, thereby avoiding the cost of retransmission. In this environment, EE is usually able to reduce outbound data streams (records) by a factor of 5 or more. The data stream that is transmitted over the radio network is thus characterized by tranmission of significantly fewer packets.

One challenge in developing data stream caching is in determining the unit (object) to be cached. Unlike other caching contexts (e.g., web browsers, CPUs) emulator data streams do not consist of a series of uniquely identifiable objects, because the emulator and the host application maintain state information and cooperate to interpret the data. The objective was to determine meaningful segments that would be large enough to get significant data reduction if they are cached and also have a high access frequency. The segmentation algorithm is a *function on the data stream* being processed -- in our case TN3270 and TN5250 data streams. In principle this *caching technology can be applied to many other data streams* by providing other segmentation functions that are specific to them.

As illustrated in Figure 2, there are two persistent caches for each emulator session: one at the SSI and one at the CSI. Making the caches persistent allows EE to adapt to the usage pattern of an individual and provide improved performance on subsequent connections. Unlike traditional buffer compression techniques, the EE persistent cache avoids having to relearn the usage pattern each time a new session is established. The caching technology is discussed in greater detail in section 4.

## Basic Flow

Let us consider typical outbound data stream. There will usually be many fields on the screen containing static text that is used to prompt for input. Some input fields may be preloaded with data that can be changed by the user. The SSI will analyze this data stream and divide it into segments. Each segment is *resolved* using the cache as follows:

A key field is generated for each segment. If a segment with this key is found in the cache then the cached segment is compared with the new segment. If the segments match, then this segment is replaced in the outbound data stream with an instruction to access the segment from the CSI's cache.

[3] The loopback feature enables a TCP/IP connection that uses only internal memory for communications.

If there is no match for the key, then the new segment is inserted in the cache. If the key is found but the input segment does not match the cache segment[4], then the existing cache segment is deleted and the input segment is inserted in the cache (with the same key). Whenever a new segment is inserted in the cache, the segment data is emitted to the output buffer with an instruction to insert the segment in the cache at the CSI. In order to avoid unnecessary wasted space the EE server will only cache segments greater than a predefined minimum size. Segments that are smaller than this minimum are replaced in the outbound data stream buffer with an instruction to copy this data without caching. The process of resolving segments generates output data stream buffers to be sent to the client (CSI).

The compression logic works on the data buffers to be sent between the SSI and CSI. Unlike caching, compression operates in both directions and starts afresh each time a new session is established. Any number of compression algorithms are effective here and the choice depends on the objectives (compression ratio versus computing cycles). We found arithmetic compression to be the most effective for these emulator data streams that contain mostly textual data. EE chose arithmetic compression[5] because, in a wireless environment, the better data reduction was deemed more important than saving CPU cycles.

The output buffer is compressed and the result is then sent over the CSI-SSI TCP/IP connection. Surprisingly, we found compression to be effective even when most of the buffer consists of segment addresses (in contrast with the actual segment data). This is because there is substantial redundancy in the segment control headers that are required to delineate segments in the data stream.

At the EE client, the incoming data stream is first decompressed. The caching instructions in the decompressed data stream are now analyzed to reconstruct the original data stream. Cached segments are retrieved and new segments stored in the cache as appropriate. In this way, the EE client maintains a mirror image of the server's view of the cache.

## 3 EMULATOR EXPRESS AND TELNET

Telnet 3270 (TN3270) [10] and later telnet 5250 (TN5250) [11] are telnet regimes that enable 3270 or 5250 datastreams, respectively, to be transported intact across an IP network via TCP/IP. With the rise of popularity of personal computers it became common to have the telnet client actually run on a PC and present the data in a window on the PC desktop. Similarly, as TCP/IP connectivity moved inside the mainframe and AS/400 themselves, the telnet (TN3270 or TN5250) server could also be implemented within the mainframe removing the need for a separate processor and communication line.

Communication between a telnet client and server is initiated by the client opening a TCP/IP socket to a port on the server. The assigned telnet well-known port is 23, although it is possible for the server to use a different port, for example if it is deemed desirable to have two different telnet servers on a system. Evidently, the user at the client end must have a method of specifying both the target host name and also the port to use. Additional configuration options are typically available to the user, including the ability to designate an SNA Logical Unit (LU) name to be used within the host system's SNA namespace.

### Sessions: Circuits of TCP/IP Connections

With the addition of the two intercepts in the system the single telnet connection, or session, is now replaced by three connections: a connection from the emulator to the CSI, a connection from the CSI to the SSI, and a connection from the SSI to the telnet 3270 or telnet 5250 server/host. The connection between the two intercepts is implemented in the present design with one TCP socket connection for each telnet session. This allows each session to be managed by its own thread or process independently of the other sessions running on the same processor. Alternate approaches might allow all sessions to be multiplexed over a single pipe between the two intercepts. Although TCP is used between the two intercepts in the present design, another protocol could be substituted if it were more suitable for the underlying transport medium. Indeed, a UDP-based protocol was tried early in the development cycle. However, the small performance gain achieved was offset by loss of reliability and less stable performance as compared with TCP.

### Telnet Protocol Reduction

In order to speed session startup some optimization of the standard telnet negotiation protocol is done between the client and server side intercepts. This is accomplished by presuming that if terminal negotiation is requested it will, indeed, be performed and then by presuming that is 3270 or 5250 terminal types and negotiated that the corresponding binary and the end of record negotiations will also be successful.

### Optimizing TCP "Keep Alive" for Wireless

An additional optimization is performed by the server side intercept which responds to keep alive messages (such as timing marks) without forcing them to the client side. Since mobile clients may frequently move in and out of range this optimization has the additional benefit of preventing session termination to a temporarily unreachable client as well as the obvious one of reducing the air-traffic. A drawback of this approach is that the server side intercept will not detect loss of connection with a client unless that loss is reported by the underlying transport mechanism. This is particularly detrimental where a client is using a particular logical unit (or LU). For example, if the client side system is restarted while out of radio range but the server does not detect the session loss then every connection attempt using the client LU name will fail until the original session is deactivated by some means or another. To address this problem the client side intercept reports what sessions are still active whenever a new session is established. This allows the server side intercept to terminate any sessions that are no longer active.

As will be described in section 4, the cloning of client IDs can cause problems. Since the server cannot distinguish between a new session from a cloned ID and an undetected restart of the

[4] The key for a segment is computed as its 32 bit CRC. Thus, it is extremely rare that two different segments can have the same key.

[5] One candidate extension for EE is to permit more than one compression option.

original owner of the ID the second system to connect will cause termination of sessions from the first system. The automatic assignment of a new client ID to one of the offending systems would be nice enhancement.

## Configuring an EE Client/Host Connection

A certain amount of configuration is necessary because the EE system uses intercept programs. The telnet emulator program is configured to communicate with the CSI rather than the ultimate telnet server. Similarly, the CSI must be configured to connect to the appropriate SSI. The SSI uses a single port to listen for incoming connections from clients. The CSI is configured with a one to one relationship between connection ports and target host systems. At the CSI one connection port is specified for each telnet server of interest and this port number maps to the IP address and port number of the target server.

The target telnet server IP address and port number are sent from the CSI to the SSI along with other configuration information used by Emulator Express. This design permits each client to define the sessions it needs without requiring any coordinated updates at the SSI.

## 4 EMULATOR EXPRESS CACHING

### Session Establishment and Cache Activation

When a session is established between a given mobile client and host application, a persistent cache is allocated at the CSI and SSI, respectively; this is called the *active* cache because it is accessed while the session is active. The first time a session is created the active caches are empty. Over the lifetime of the session, segments are stored and retrieved from the active cache; the cache becomes increasingly populated and the degree of data reduction increases as there will be more “cache hits.” For caching to work correctly, it is necessary that the active cache on the CSI and SSI remain perfectly synchronized; i.e., if the SSI replaces a data segment with a cache reference, it is necessary[6] to guarantee the availability of the cached entry at the CSI so that the segment data can be retrieved.

For subsequent sessions, however, we would like to use a cache that was generated during a previous session between the same client and host. To reuse caches across session instances requires that the active caches be saved with complete integrity before the session terminates. However, what if the session is abruptly terminated due to loss of signal, network outages or a power loss? Abrupt disconnects are normal in a wireless environment. Following a sudden failure, the active cache must be considered useless because its state is unknown; segments in-transit or in memory buffers will be lost. To guarantee the generation of consistent versions of the cache, a checkpointing protocol was developed.

### Checkpointing

To cope with the prospect of unexpected failures, it is necessary to periodically generate a consistent copy of the active cache, called a *checkpoint*. The criteria for when to take a checkpoint is a function of session activity and time. For example, if the checkpoint time interval has passed but no activity has occurred on the session, no checkpoint is taken.

#### *Taking a Checkpoint*

When the CSI determines that a checkpoint is necessary, it calls the cache manager to *prepare* the active checkpoint. The active cache files are locked, dirty pages are flushed to disk, the active cache files are copied to a temporary files, the cache state is set to *prepared,* and finally the active cache files are unlocked so that session activity may continue.

After the CSI's cache is prepared, a checkpoint request is sent to the SSI; this command is “piggy backed” on the next message sent to the host to avoid an extra message flow. The SSI first prepares its cache in the same manner as the CSI and immediately commits[7] to create the next cache checkpoint. At any given point in time there may be two checkpoints labeled CP0 and CP1. This is necessary to ensure that there is always a valid checkpoint even if a failure occurs during checkpoint creation. After the checkpoint is successfully created, the SSI returns a positive acknowledgment to the CSI consisting of the checkpoint time stamp and a checkpoint number (0 or 1). As with the checkpoint request, the acknowledgment message is piggy-backed on the next message sent from the host to the client. When the CSI receives the positive acknowledgment, it creates its next checkpoint in like manner as the SSI except that it tags its checkpoint with the SSI's checkpoint time-stamp and checkpoint number received on the checkpoint response. The CSI also sets state information indicating that a confirmation response to the SSI is pending. A confirmation message is sent from the CSI to the SSI to confirm successful generation of the checkpoint on the client device. Now

Hp is the session listening port number for the host connection. However, this scheme fails to take into account that dynamic address assignment (DHCP) is typically used to assign IP addresses for mobile client devices[9], rendering C useless for uniquely identifying the client device across modem connections (wireless or wireline).

### *Unique Client Identifiers*

To overcome this problem a unique[10] persistent *ClientID* is generated by the SSI the first time a client device connects. When a session is initiated, the CSI sends a *SelectCheckpoint* command. Initially, it contains a null ClientID. When the SSI receives the null ClientID, it generates a unique identifier and returns it to the CSI. ClientIDs are made persistent at both ends. At the SSI the ClientID is used as an anchor to locate all the checkpoints for all sessions initiated by a specific client device. On subsequent connections to the same SSI, the SelectCheckpoint command will contain a non-null ClientID in addition to the time-stamp and checkpoint number of the latest checkpoint. These data enable the SSI to locate and activate the matching checkpoint for the session.

If a clientID received by the SSI does not exist, the SSI reuses the ID; the server does not allocate a new ID. The current client design does not have any method for the client to replace an existing ID automatically. This has causes a slight inconvenience in cases where a mobile rollout is done by cloning hard drives on mobile systems. A necessary, but often forgotten, step is to ensure that the stored client ID from the copied system is removed from the new copy.

### *Activating a Checkpoint*

During session startup, the CSI activates its most recent checkpoint with the expectation that the SSI can do likewise. Activating a checkpoint means that the checkpoint files are copied to temporary files that will serve as the active cache during the life time of the session. The checkpoint files themselves are never modified once created. Next, the CSI sends a SelectCheckpoint command to the SSI and sets its state to *checkpoint response pending*. Upon receipt of a SelectCheckpoint command, the SSI attempts to locate the checkpoint identified by the time-stamp and checkpoint number in the command. Theoretically, by design of the checkpointing protocol, the SSI should always find the checkpoint specified in the SelectCheckpoint command. If found, a positive response is sent to the CSI. If, for some reason (e.g., media failure) the checkpoint is not found, the SSI returns a "NotFound" response to the CSI, and creates a fresh active cache. When the CSI receives the NotFound response, it also starts with a fresh active cache, and processing continues. Eventually, checkpoints will occur to generate new checkpoint instances.

## Multiple Sessions to the Same Host

On VM and MVS systems some users may have more than one user ID and, therefore can have multiple concurrent sessions with the same host. With AS/400 systems, users may have multiple sessions for the same user ID. With modern emulators these sessions appear in separate client windows. Therefore, it may be quite natural and productive for a user to interactively switch among various host applications by clicking different emulator windows on the desktop.

This requirement posed some interesting design challenges for EE. Does each such session have its own cache and checkpoints? How are these sessions identified and bound to the proper cache? Must the checkpoint activity be synchronized across the different sessions? Are any additional configuration parameters needed?

All sessions between the same client/host that have the same CSI listening port share common cache checkpoint files. However, each session gets its own active cache when the session is started. Checkpointing on each session is done independently. The last session to execute a successful checkpoint creates the latest checkpoint. However, since the checkpoint files are shared, it is possible for a checkpoint request to be issued on one session while a checkpoint is in progress on another. One of the goals in our design was to never require one session to wait for the completion of a checkpoint in-progress on another session. In a wireless environment this could take seconds.

Race conditions can occur either when one session is being established and another is checkpointing or more than one sessions are attempting to checkpoint at the same time. The only time a checkpoint is not available for activation is when it is being created. The CSI and SSI allow for two checkpoints to exist: CP0 and CP1. Persistent state information is maintained that records the checkpoint states for a given ClientID. A labeling scheme is used to guarantee that at least one checkpoint is available for activation. Only one checkpoint is permitted to be in-progress at any given time ensuring that at least one of the two checkpoints will be free (unlocked) at all times.

If CP0 is the oldest checkpoint, then the next checkpoint request will result in the construction of a new CP0 (CP0') and CP1 will be marked as the oldest checkpoint. CP1 will be activated as the active cache for any new sessions that are started during the checkpoint processing. The files associated with CP0 are deleted when the prepared cache is committed. At the CSI after the positive acknowledgment is received from the SSI and the new checkpoint is committed, the files associated with CP1 are deleted. At the SSI the files associated with CP1 are deleted when a confirmation message is received from the CSI as described previously. It is necessary to retain the CP1 files until the confirmation is received in order to guarantee that there is a valid checkpoint pair at the CSI and SSI even in the event of checkpoint failure.

### *Checkpoint Races*

If a CSI attempts a checkpoint while one is in progress for another session[11] or the checkpoint is being activated for session start-up, the checkpoint request is simply aborted and normal processing continues. This polite, laissez-faire philosophy toward checkpointing is based on the recognition that the exact timing or

[9] Using the TCP/IP PPP or SLIP protocols to connect to user end-point devices.

[10]Currently the ClientID is a 4-byte entity composed of the 3 low order bytes of clock time and a one byte sequence number. For multiple server (SSI) support, a 16 byte GUID could be used.

[11]To the same host using the same IP address and port number

even the contents of a checkpoint is not crucial as long as the checkpoints are created with reasonable frequency. What is crucial is that the checkpoint protocol guarantee that the checkpoints created at the CSI and SSI are perfectly synchronized and that the protocol does not introduce noticable delays in normal session traffic. In all likelihood different sessions will win the checkpoint race over a reasonable period of time.

### *Observations*

The above design has the virtue of simplicity, non-blocking, and continuous availability. It is guaranteed that session initiation is always possible, very few resources are locked for very long and no waiting is required for cache checkpointing. In the event of failure to activate a checkpoint, processing starts from scratch with empty caches. However, at first glance it seems to have one major deficiency. Namely, if we assume that different sessions are dedicated to different host applications, one might expect that the cache build up for one application would not be effective for use by other concurrently running applications. In the above design, the latest checkpoint would reflect the activity of the application running on a given session.

The above concern has not proven to be a problem in practice. This may be because there are lots of common segments across disparate applications (e.g., logon screen). In any case there is a simple prescription that users can follow to obviate the problem. If it is important for an application to have its own dedicated cache, the user can simply specify unique listening port for the application.

Assuming a common listening port for all applications to the same host, if a user executes his/her suite of common applications sequentially, then the effect will be to form the union of their respective caches. This might be best done in a wireline modem or LAN environment to *prime* the cache. Subsequently, when multiple sessions to the same hosts are constructed in a wireless environment, the cache will contain a representative set of segments for all the applications. Over time the cache will become biased toward the most frequently used application -- a desirable effect.

## The Cache Manager

The cache manager is the software component that allocates and deallocates cache space and stores and retrieves variable length segments to and from the cache. This cache had to meet a number of requirements:

- Ability to store variable length segments (from 16-4000 bytes)
- Efficient random keyed access to the data: Each segment is typically small (e.g., 100 bytes). It is important to quickly access a given segment as a function of its key and minimize disk I/O, particularly for frequently referenced segments.
- Bounded in size and self-organizing: Since a session may run an indeterminate length of time, a means to reclaim space had to be provided when the cache becomes full. A least-recently-used (LRU) algorithm was implemented to delete the oldest segments when the cache becomes full.
- Persistent across sessions: The cache data had to be persistent so that the benefits accrued from caching on one session (from a given client/SSI/telnet server) could be carried over to the next session.

An EE cache consists of three files: a *segment store* and an *index store*, and a *state file*. Each store is comprised of a set of fixed-length blocks. Paging is used keep as many of these blocks in memory as possible. Our measurements show that on the order of 20% of the segment accesses require file I/O. The state file pertains to cache checkpointing discussed above.

The *SegmentCache*, supports the ability to store variable-length data segments using one or more blocks of the segment store[12].

The *index store* is implemented as a B-Tree where each block corresponds to a B-Tree node. Each B-Tree entry is composed of a segment key and the address of the segment in the segment store. The segment key is a 32-bit CRC value computed for its respective segment. The address of a segment is the block number of the first block of a segment in the segment store.

The CSI and SSI components access the cache via an interface that *resolves* segments. The interface guarantees that the segment store and index store are always consistent.

## 5 PERFORMANCE

New users are often amazed at the responsiveness of applications when using Emulator Express. Comments such as “this is faster than my office LAN” are not uncommon, even though the user is using an application from a car-mounted mobile terminal.

While the actual reductions seen for different application sets vary widely, ratios of between 5:1 and 10:1 are common over a complete session with occasional figures as high as the 25:1 measured above for individual screens with high amounts of repetition. Note also that inbound data from CSI to SSI may slightly increase where input is mostly an attention key with no user data input. However, the resulting data is still much smaller than outbound data and any significant user input will generally cause compression to have some real effect on inbound data.

Response time improvements due to the data reduction are significant. For example, in one suite of test runs, the average response time for a native dial-up 2400 baud connection was over 11 seconds compared with less that 2 seconds for the same tests and connection speed using Emulator Express.

In general, many factors can influence the performance of Emulator Express systems, including the speed and reliability of the connection between the client and the server systems. Other factors include the nature of the workload or application, the duration of sessions, the SSI load, the network load between the SSI and the host or telnet server and the host load to name a few. Therefore, while the performance data on Emulator Express has been very impressive, actual performance seen by different Emulator Express installations may vary.

[12]In OO terminology, a segment store is an *instance* of the segmentCache class.

## 6 CONCLUSIONS

This paper has described novel technologies that make it possible to run general 3270 and 5250 emulation from a mobile unit over very low bandwidth wide area wireless networks. These technologies (data stream caching and telnet protocol reduction) combined with traditional compression, are describe in the context of their implementation in IBM's Emulator Express, part of IBM's eNetwork wireless product suite. Performance measurements and customer experience show that Emulator Express enjoys a significant compression ratio for telnet traffic (commonly 5:1 up to 10:1) which reduces the cost and bandwidth required for delivering applications to mobile workers. It enables existing applications to be deployed in a mobile environment using relatively slow radio networks for communication by improving response times enough to match those achieved using common wireline speeds.

Perhaps the most far reaching aspect of this work is the development of the data stream caching technology. In principle, this technology can be applied to venues other than emulation and, therefore, can contribute toward improving efficiencies for a wide range of distributed network applications.

## REFERENCES

[1] *IBM eNetwork Wireless Software.* Product documentation on the IBM eNetwork wireless software products including eNetwork Wireless Client and Gateway and eNetwork Emulator Express can be found at URL: *http://www.software.ibm.com/enetwork/mobile/library*

[2] Host On-Demand Certified "100% Pure Java", IBM Host On-Demand Certified 100% Pure Java. Web-to-Host Access Software Runs Smoothly on All Java-Enabled Systems. URL: *http://www.networking.ibm.com/netmsg22.html*

[3] *CICS Transaction Gateway, Version 3.0, Administration and Programming*, IBM SC34-5448-00 , September 1998. URL: *http:// www.software.ibm.com/ts/cics/library/manuals*

[4] Tomasz Imielinski and B. R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," CACM (37,10), October, 1994.

[5] *ARTour Emulator Express Server for AIX, Version 2*, March 1996, IBM GC31-8299-00.

[6] *ARTour Emulator Express Server for OS/2, Version 2*, March 1996, IBM GC31-8298-00.

[7] Barron Housel and David Lindquist, "Web Express: A System for Optimizing Web Browsing in a Wireless Environment," Proc. MobiCom '96.

[8] H. Chang, et al., "Web Browsing in a Wireless Environment: Disconnected and Asynchronous Operations in ARTour Web Express," Proc. MobiCom '97, Sept. 1997, pp260-269.

[9] *RFC 854: Telnet Protocol Specification*, IETF Network Working Group, J. Postel, J Reynolds, NIC 18639, May 1983.

[10] *RFC 1041: Telnet 3270 regime option*, IETF Network Working Group, P. Rehkter, January, 1988.

[11] *RFC 1205: 5250 Telnet interface*, IETF Network Working Group, Chmielewski, P., February, 1991.