

Mimic: Raw Activity Shipping for File Synchronization in Mobile File Systems

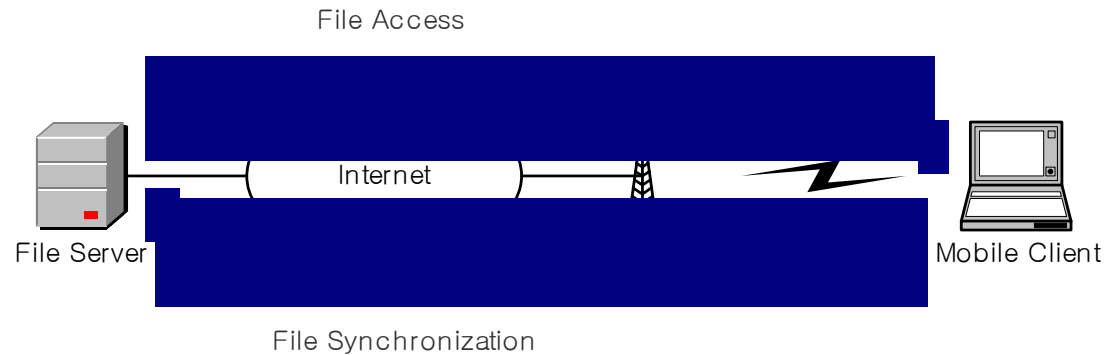
Tae-Young Chang, Aravind Velayutham, and
Raghupathy Sivakumar

School of Electrical and Computer Engineering

Georgia Institute of Technology

Atlanta, GA 30332, USA

File Synchronization in Mobile File Systems



- File synchronization in distributed file systems is a consistency restoration process between a file server and a client
 - Over mobile networks such as a WWAN, file synchronization in traditional file systems cannot be performed effectively due to limited bandwidth
- Bandwidth usage efficiency of file synchronization is important

Bandwidth-Efficient File Synchronization (1)

- Data compression scheme
 - Compresses each block of data or meta-data
 - *On-line data compression in a log-structured file system* [Burrows'92]
- Differential update scheme
 - Exploits similarities between versions of the same file
 - Based on the *diff* scheme of the UNIX systems
 - *Rsync* [Tridgell'00]
 - *Low-bandwidth network file systems* [Muthi'01]

Bandwidth-Efficient File Synchronization (1)

- Operation shipping
 - Logs and ships **user operations** that update the files
 - Session/application level logging and replaying
 - Need modification for GUI-based interactive applications
 - Corrects replaying errors by **forward error correction (FEC)**
 - Minor re-execution discrepancies caused by non-repeatable operations can be detected by fingerprint algorithms
 - *Operation shipping for mobile file systems* [Lee'02]

Motivation

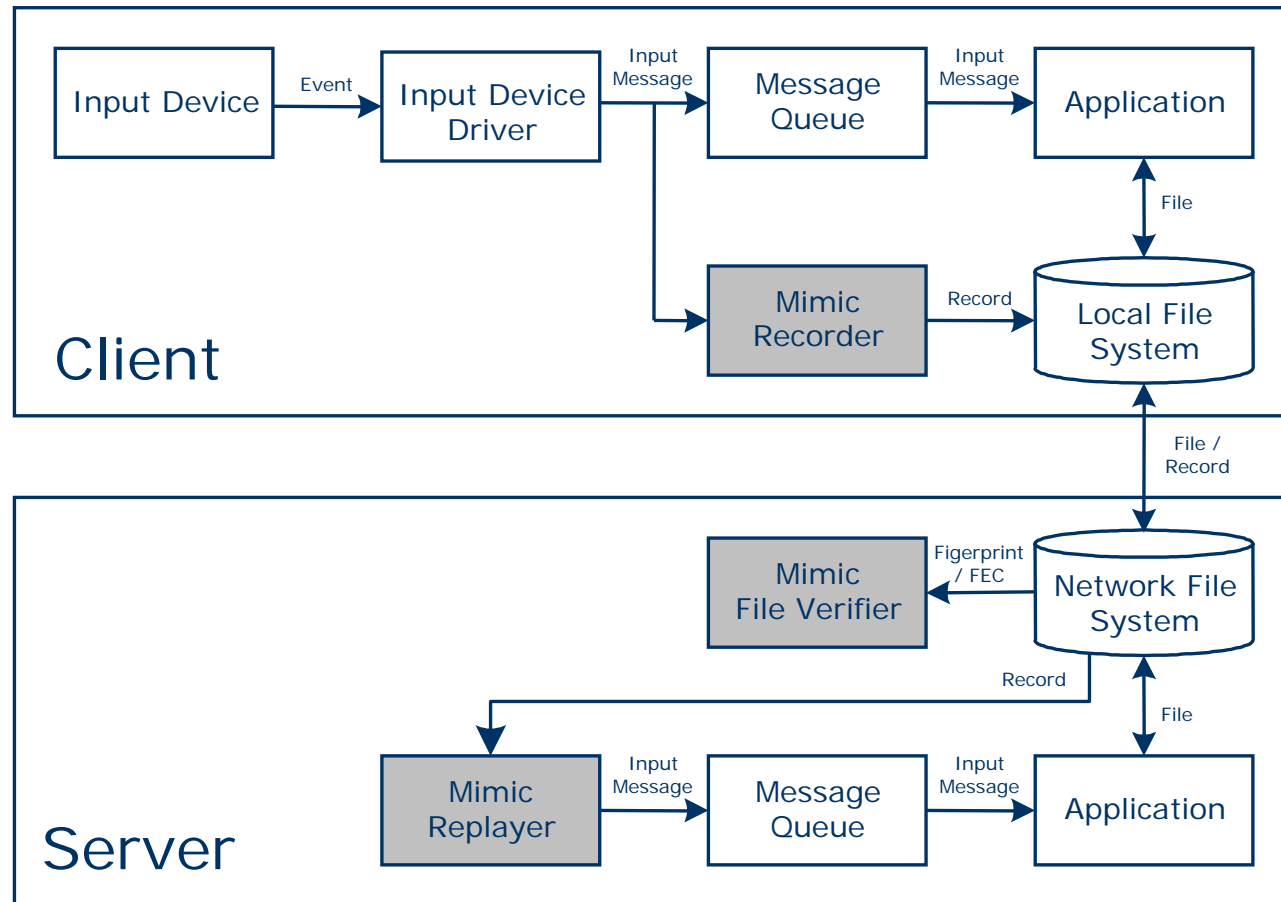
Update size comparison in Microsoft Word

Activity Description	Activity Events	Full File Size	<i>diff</i> Size	Activity Size
Insert a line	98 keystrokes	29341 B	1543 B	236 B
Insert a paragraph	476 keystrokes	29356 B	2111 B	848 B
Copy and paste a paragraph from the same file	6 keystrokes + 12 mouse-clicks	33449 B	1119 B	72 B
Change the font type of a paragraph	7 mouse-clicks	40611 B	1660 B	30 B

Goal and Overview

- Goal of our work
 - To design an application-unaware activity shipping scheme for file synchronization in interactive applications
- Mimic is a file synchronization strategy that records raw activity at the client side, ships the records to the server during synchronization, and replays the activity at the server

Mimic System Overview

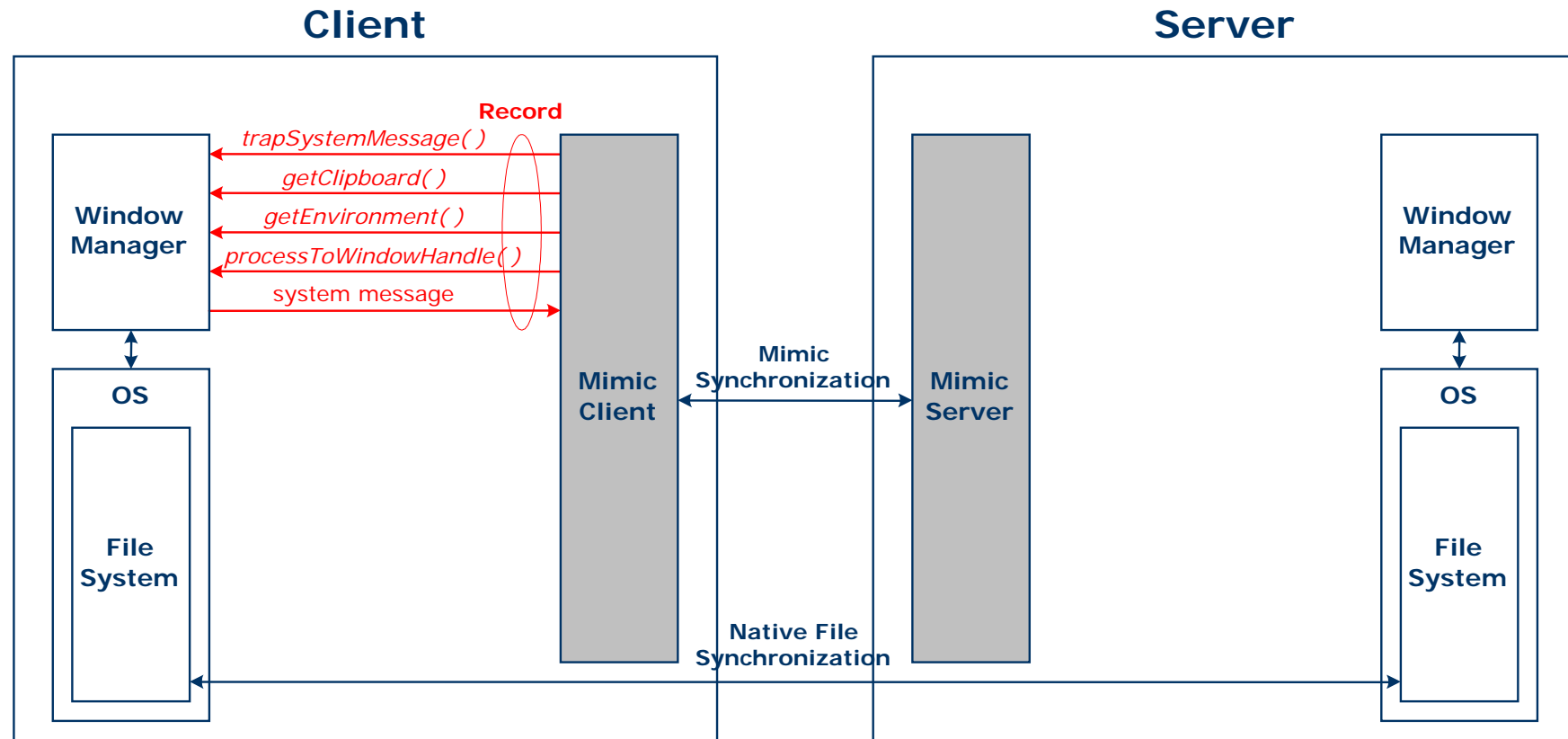


Mimic Design Elements

- Raw activity recording
 - Mapping filename to process/session
 - Record optimization
- Raw activity replaying
 - Environment synchronization
 - Replaying speed optimization
- Integration of Mimic with file systems
- Verification/error correction of replayed files
 - Presented in [Lee'02]

Mimic Client Design:

Mimic Client to Window Manager Interface



Window Handle Table (WHT)

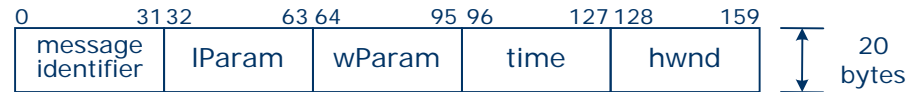
- WHT maps the process (or session) handle and filename of an application to the corresponding set of window handles
 - When a file is opened with a filename, the mapping information is acquired, and added on WHT
 - through *processToWindowHandles()*
 - When the file is closed, the mapping is removed from WHT
- Mimic recorder captures system messages having the window handles listed on the WHT
 - through *trapSystemMessage()*

Descriptors

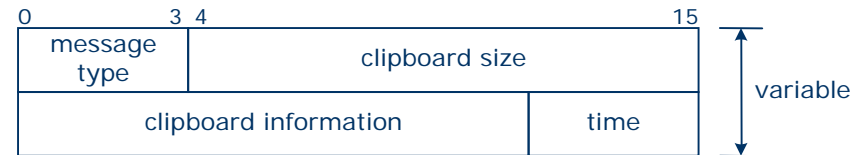
- Each captured message is translated into a descriptor
- **Activity Descriptor (AD)**
 - Describes an individual user input activity for an application
- **Meta Descriptor (MD)**
 - Capture the changes of system environment during recording
 - Includes screen resolution, color depth, keyboard layout, clipboard content, etc.
- **Environment Descriptor (ED)**
 - Describes the initial system environment when recording begins
 - Same structure as that of an MD
 - through *getEnvironment()* and *getClipboard()*

Mimic Client Design:

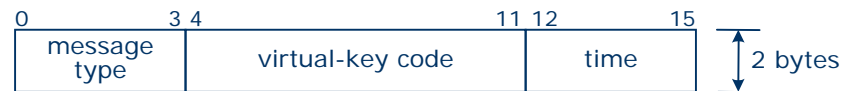
Descriptor Structures



EVENTMSG in Windows



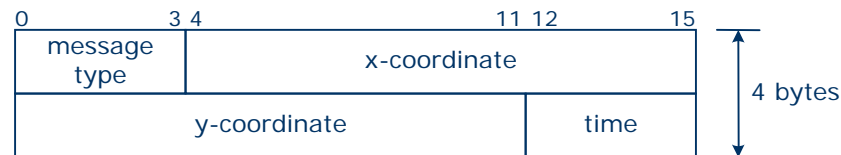
Meta Descriptor (Clipboard Content)



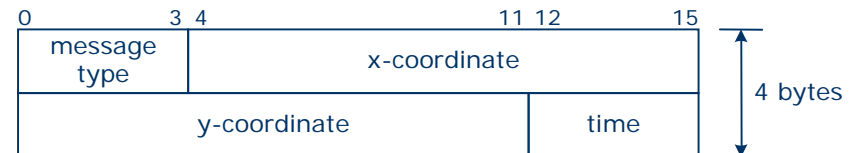
Activity Descriptor (Keyboard Activity)



Meta Descriptor (Keyboard Layout Change)



Activity Descriptor (Mouse Activity)



Meta Descriptor (Screen Resolution Change)



Activity Descriptor (Meta Descriptor Link)



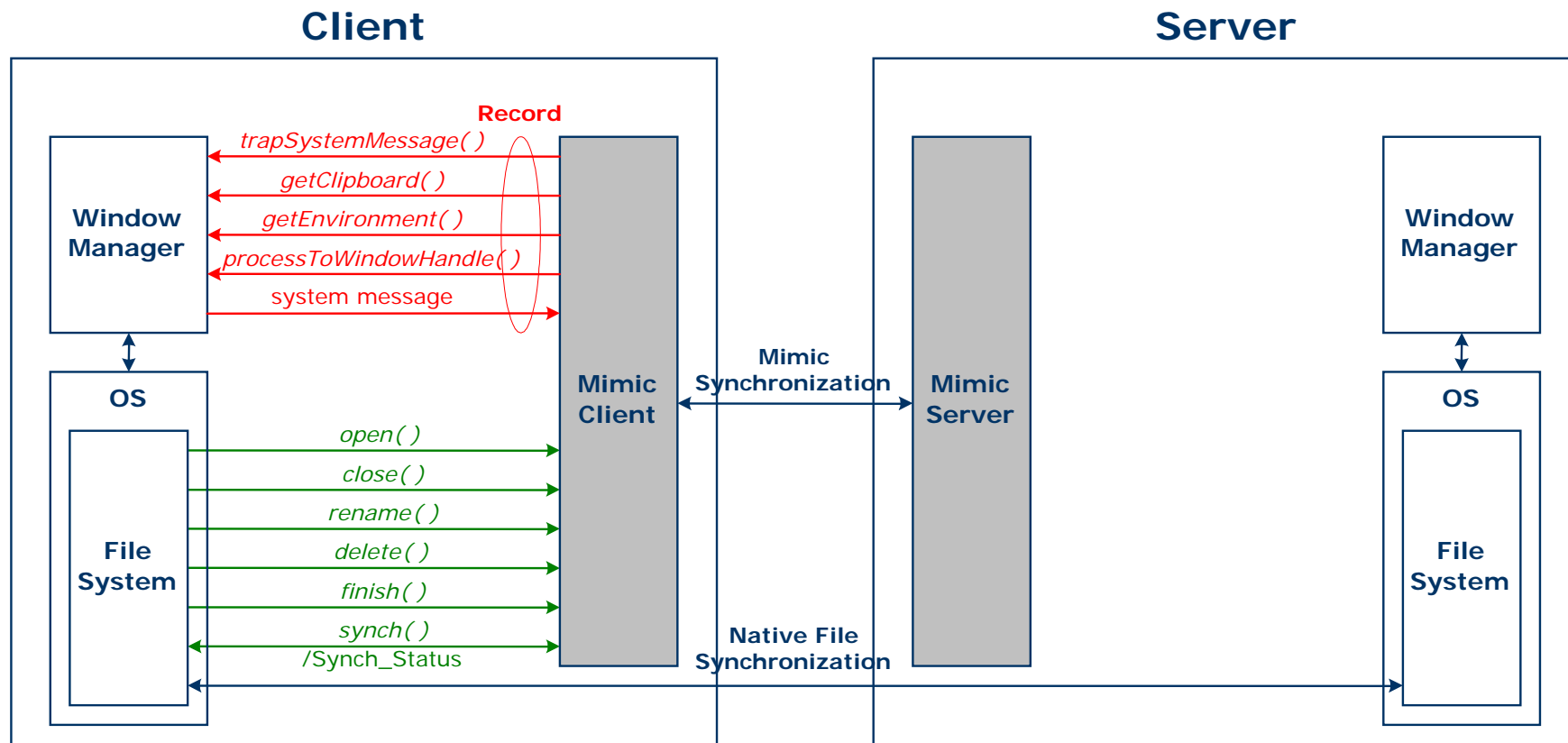
Meta Descriptor (Color Depth Change)

Records

- Each descriptor is recorded in a record
- **File Activity Record (FAR)**
 - Maintained per file
 - Consisting of file information, a set of environment descriptors (EDs), a sequence of activity descriptors (ADs), and all other information
 - Linked to a meta descriptor (MD) of a meta activity record (MAR) when an environment change happens
- **Meta Activity Record (MAR)**
 - Shared by file activity records (FARs)
 - Consisting of a sequence of meta descriptors (MDs)

Mimic Client Design:

Mimic Client to File System Interface

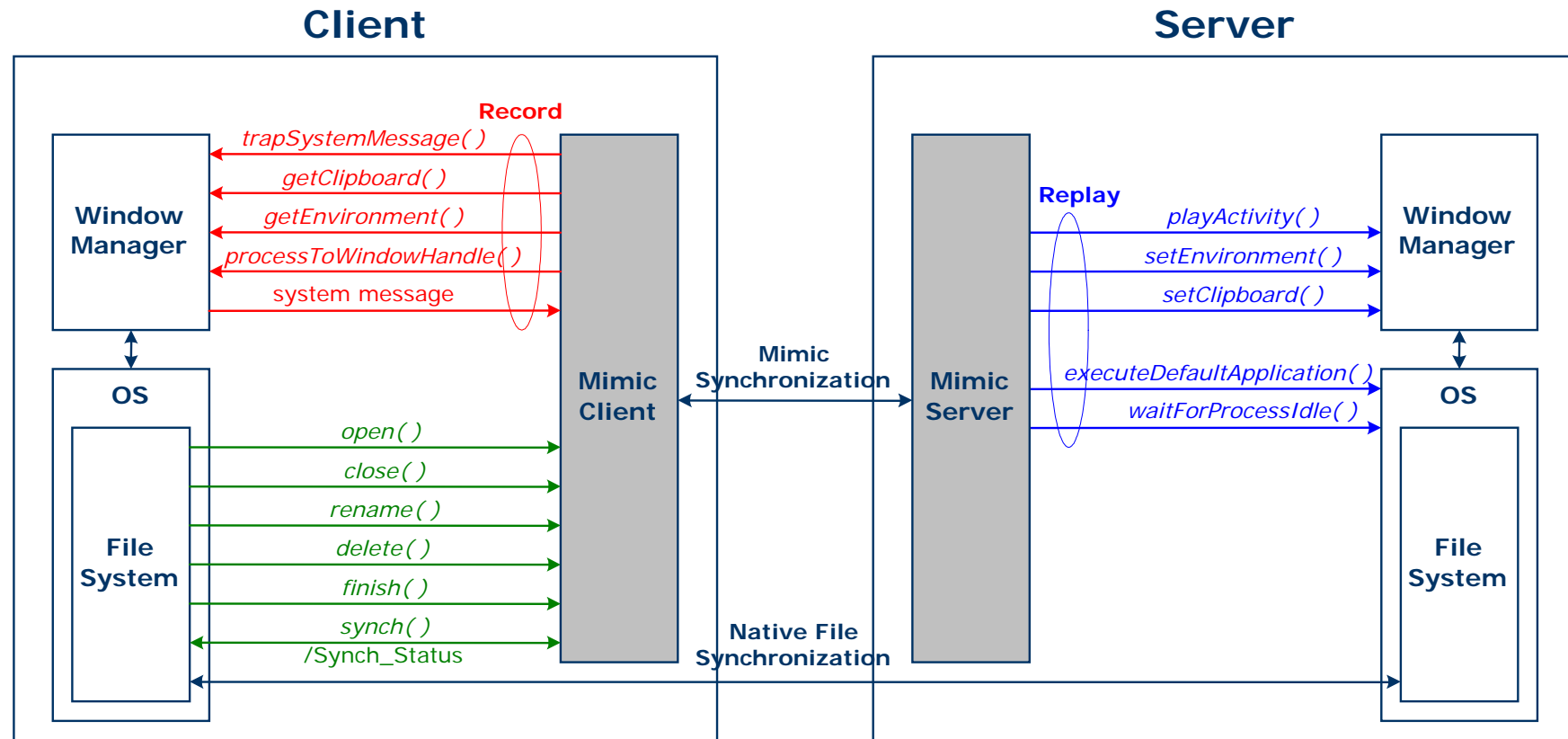


Integration with File Systems

- *open(filename, mode, process_handle)*
 - Called when a shared file is opened
- *close(filename), rename(filename), delete(filename)*
 - Called when a shared file is closed, renamed, or deleted
- *synch(filename, diff_size)*
 - Called when a shared file needs to be synchronized
 - Decides update mode by comparing FAR_size with *diff_size*
 - Returns Synch_Status
 - SYNCH_FAIL if Mimic synchronization is failed or *diff* is chosen
 - Updates again on *diff* mode when Mimic synchronization failed
- *finish()*
 - Called when the synchronization process is completed

Mimic Server Design:

Mimic Server to Window Manager/OS Interface



Initialization

- Environment synchronization
 - Based on the environment descriptor (ED) of the FAR
 - Initial system environment synchronization
 - through *setEnvironment()*
 - Clipboard content synchronization
 - through *setClipboard()*
- Application synchronization
 - Opens a corresponding application of the file activity record
 - Based on the file extension of the filename
 - Sets the same environment such as window size and location
 - through *executeDefaultApplication()*
 - Moves the system focus to the application

Replaying

- System message/function generation
 - Activity descriptor (AD) is mapped into an input system message
 - Meta descriptor(MD) is mapped into a system function to set the environment
 - through *playActivity()*
- System message/function re-execution
 - Deliver the messages to system message queue
 - Run the system functions

Replaying Speed Control

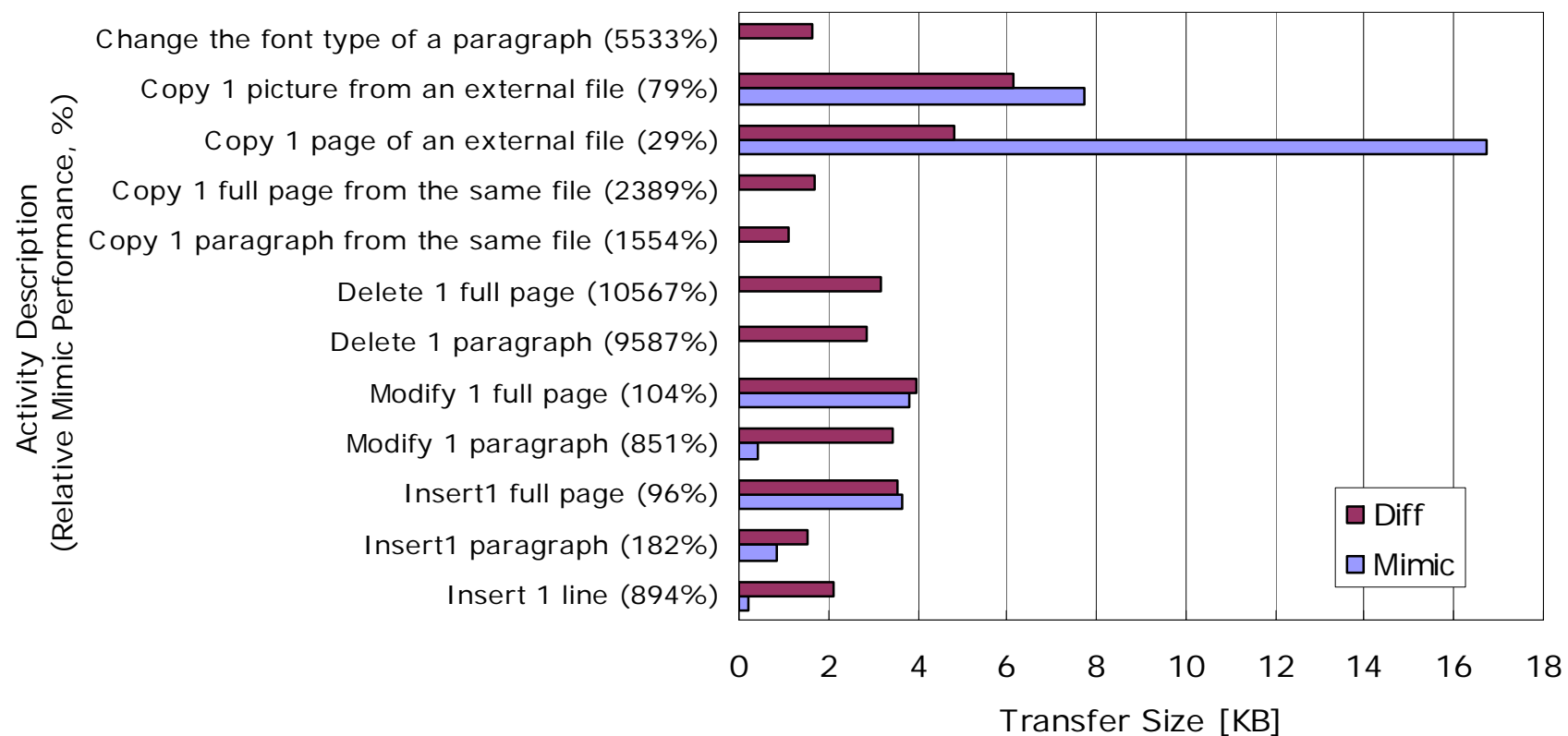
- User activity skipping and misinterpretation
 - Certain inputs are relevant to the application **only for particular states**
 - **Too fast replaying** may not let the application wait for a particular state, and cause replaying errors
- Replaying speed control in Mimic
 - Monitors the CPU utilization of the process after every message playback
 - **Plays back the next AD, only when the process is idle**
 - through *waitForProcessIdle()*

Experiment Setup

- Wireless wide area network (**WWAN**)
 - CDMA2000-1X cellular network
 - Effective data rate: about 17 Kbps
 - Round-trip time between the client and server: about 300ms
- Operating system/application
 - Microsoft Windows 2000 Professional
 - Microsoft Office 2000 suite
- Metrics
 - Transfer size
 - Synchronization latency
 - Includes shipping, replaying, and verification delays
 - Compared with the differential update (*diff*)
 - xDelta for Windows

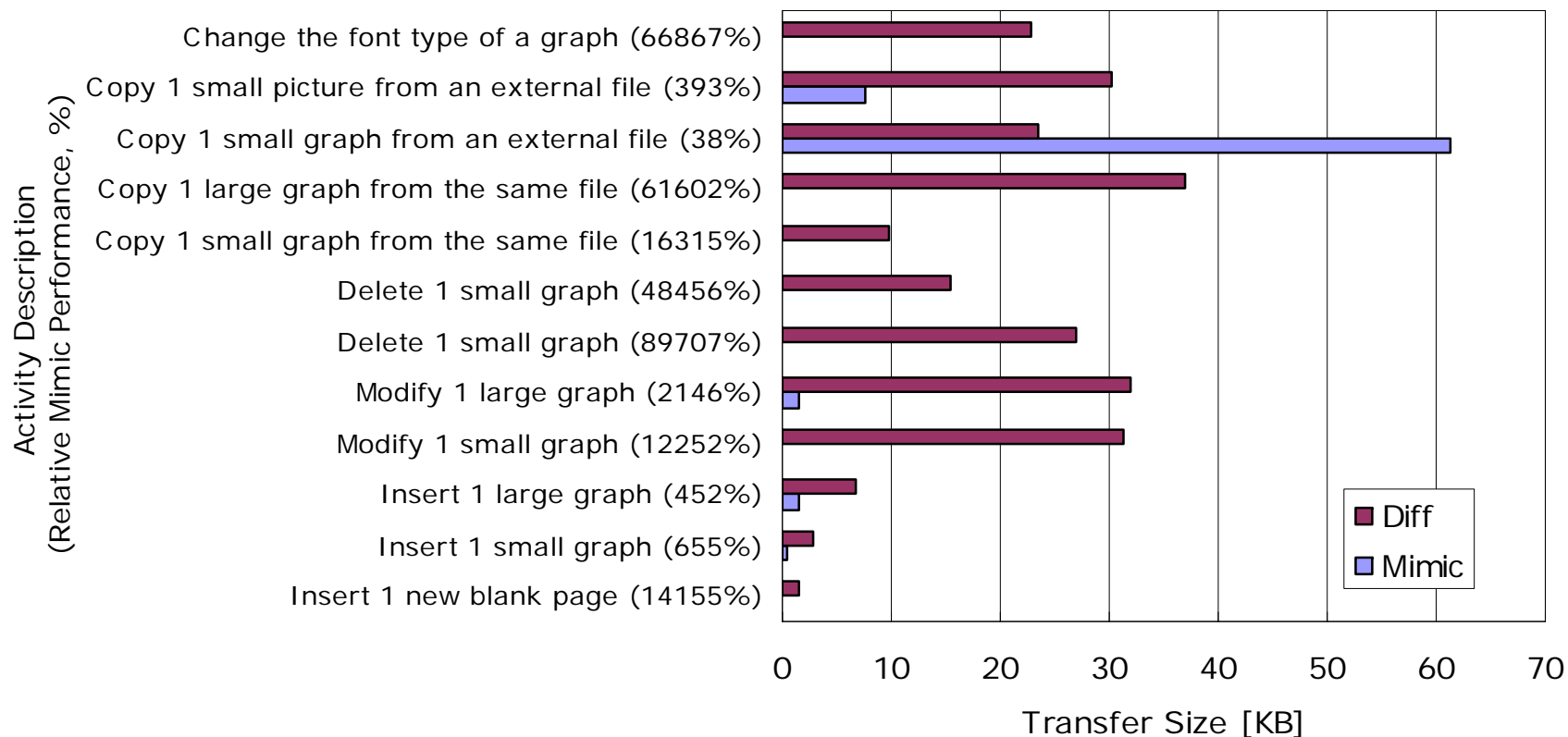
Transfer Size Results (1)

Transfer Size for Microsoft Word



Transfer Size Results (2)

Transfer Size for Microsoft Visio

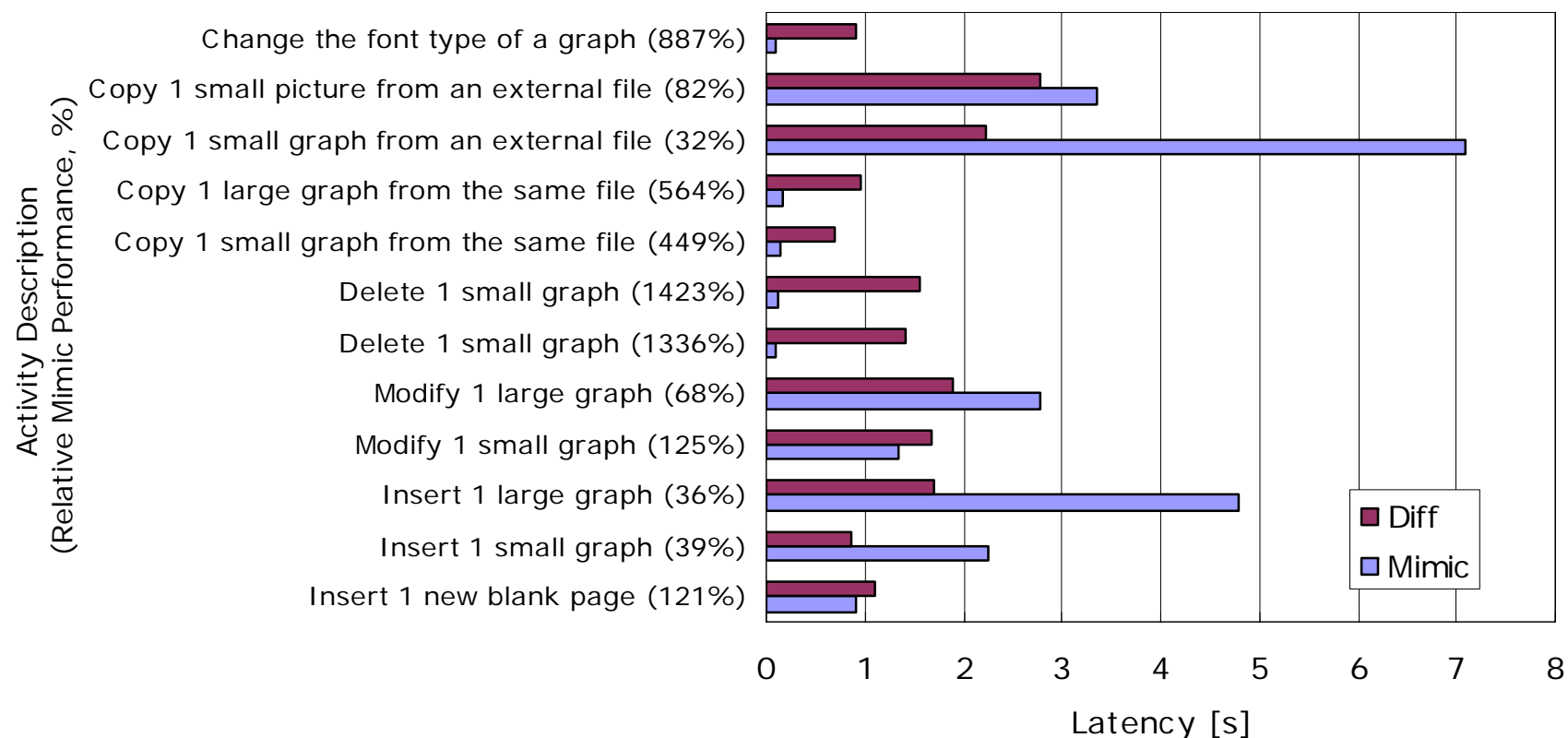


Transfer Size Results (3)

- Transfer size in Mimic is generally equal or less than that in the differential update
 - Except when copying from outside the file
 - Due to bandwidth-inefficient clipboard structure
- Mimic overhead is generally proportional to activity size
 - Except when copying from outside the file
 - Transfer size relies on the size of the copied object
 - *Diff* overhead is not proportional to activity size
 - Single line insertion in *diff* may consume more bandwidth than a single paragraph insertion

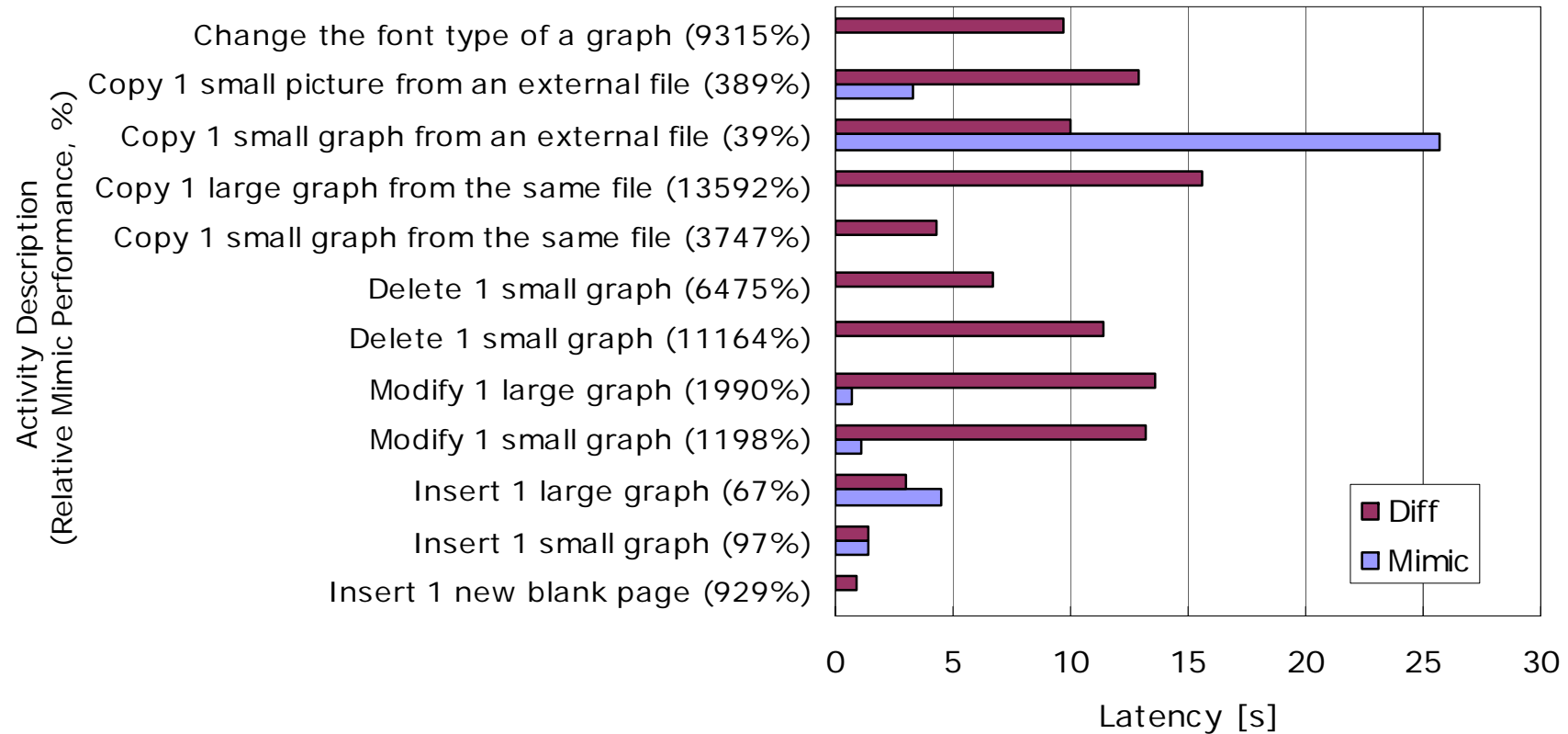
Synchronization Latency Results (1)

Latency for Microsoft Word



Synchronization Latency Results (2)

Latency for Microsoft Visio



Synchronization Latency Results (3)

- Mimic still shows better latency performance for certain activity or applications
 - Even though the latency in Mimic includes its playback time, its total update time does not exceed that of *diff*
 - Benefit of small transfer size for those operations is larger than playback overhead
- However, for the other types of activities, the differential scheme performs better in terms of latency
 - For large insertions and external copies

Conclusions

- We propose **an application-unaware approach called Mimic** that depends on transferring raw user activity records to the server, where the file is updated through a playback of the raw user activity on the old copy of the file
- We show that Mimic performs much better than *diff* in most scenarios in terms of the transfer file sizes
- We conclude that Mimic can be **used in tandem with *diff*** to substantially improve file synchronization performance