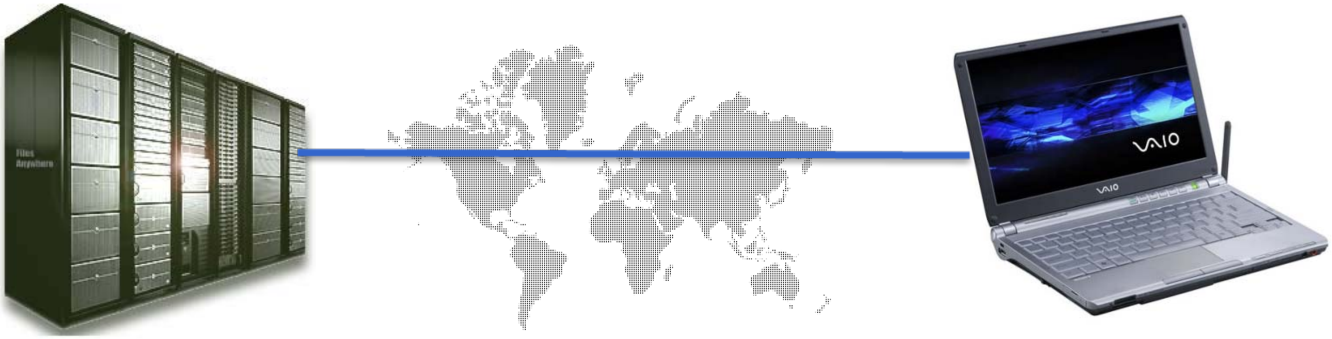
On the performance of wide-area thin-client computing

A. Lai and J. Nieh, Columbia U.

ACM TOCS, May 2006

Thin-client computing

- Client and server communicate over a network using a remote display control
  - Client sends user input, server returns screen updates
- Graphical display can be virtualized and served across a network to a client device
- Application logic is executed on the server



Key question

- Technology enablers
  - Improvements in network bandwidth, cost and ubiquity
  - High total cost of ownership for desktop computing
- Big business opportunity
  - Sun Microsystems, Google, Microsoft, …
  - Citrix MetaFrame, AT&T Virtual Network Computing, …
- *The effectiveness of thin-client computing over the wide-area network is unclear*

Goal and experimental design

- Compare thin-client systems to asses basic display performance and feasibility in WAN
- Platforms evaluated
  - Citrix MetaFrame 1.8, Windows 2000
  - Windows 2000 Terminal Service
  - Tarantella Enterprise Express II, Linux
  - AT&T VNC v3.3.2, Linux
  - Sun Ray I, SunOS
  - XFree86 3.3.6 (X11R6), Linux
- Parameters
  - Encoding of display primitives
  - Client pull/Server push and lazy/eager screen updates
  - Compression used for screen updates
  - Max display color depth supported by the platform
  - Transport protocol

Thin-client platforms characteristics

Platform	Display Protocol	Display Encoding	Screen Updates	Compression	Max Display Depth	Transport Protocol
Citrix MetaFrame	ICA	Low-level graphics	Server-push lazy	RLE	8-bit color	TCP/IP
Microsoft Terminal Service	RDP	Low-level graphics	Server-push lazy	RLE	8-bit color	TCP/IP
Tarantella	AIP	Low-level graphics	Server-push, eager or lazy based on bwd load	Adaptively enabled, RLE, and LZW at low bwd	8-bit color	TCP/IP
AT&T VNC	VNC	2D draw primitives	Client-pull, lazy upd. bet/ client requests discarded	Hextile (2D RLE)	24-bit color	TCP/IP
Sun Ray	Sun Ray	2D draw primitives	Server-push eager	None	24-bit color	UDP/IP
X11R6	X	High-level graphics	Server-push, eager	None	24-bit color	TCP/IP

Measurement methodology

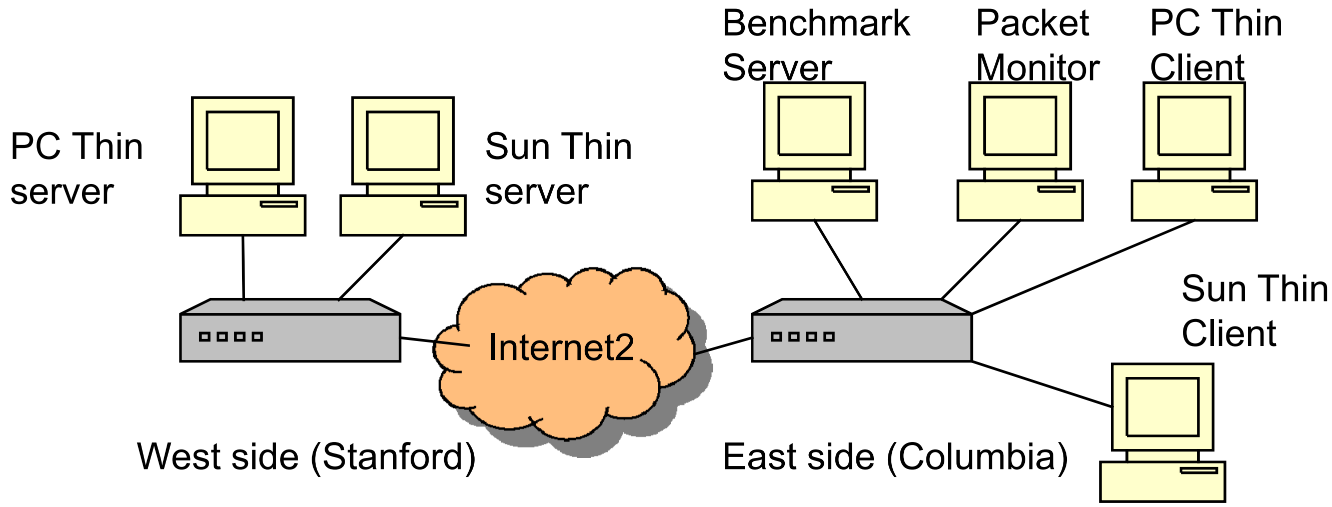
- Standard app benchmarks measure server performance, but client experience may be quite different
  – Output display may be completely decoupled (display updates merged, packets dropped, etc)
- You can't just instrument clients – most thin-client systems are proprietary and closed-sourced
- Solution – slow-motion benchmarking
  – Use packet monitor on client side
  – Latency of an operation – from first client packet sent to last server packet received
  – Introduce delays between separate visual components to isolate exchanges, later combine them to get overall results

Measurement methodology

- Slow-motion's limitations and risks
  - Does not include time from when client receives update to when the image is drawn to the screen
    - Particular an issue if client and server are not of comparable performance
  - Potential problems with TCP
    - Delays may reset TCP congestion window to initial values and force going through slow-start
    - Delays may avoid Nagle impact by allowing the client to receive all acks
- Mostly avoided or measured
  - Client and sever with comparable performance
  - Impact on TCP for one application – reported at 1-10%

Experimental testbed

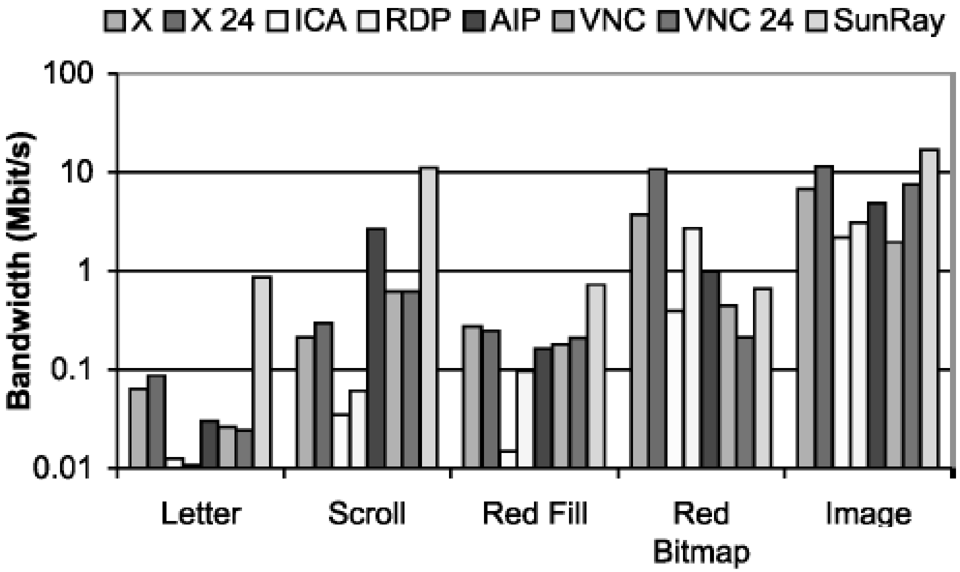
- Two pairs of thin-client/server systems, packet monitor server and web server
- Network values
  - Minimum available bandwidth from East-West: 100Mbps
  - Measured ping ~66.35ms
  - TCP window size used 1MB
  - iperf measured available bandwidth 45Mbps
- Internet2, simulated Internet2 and 100Mbps LAN



Application benchmarks

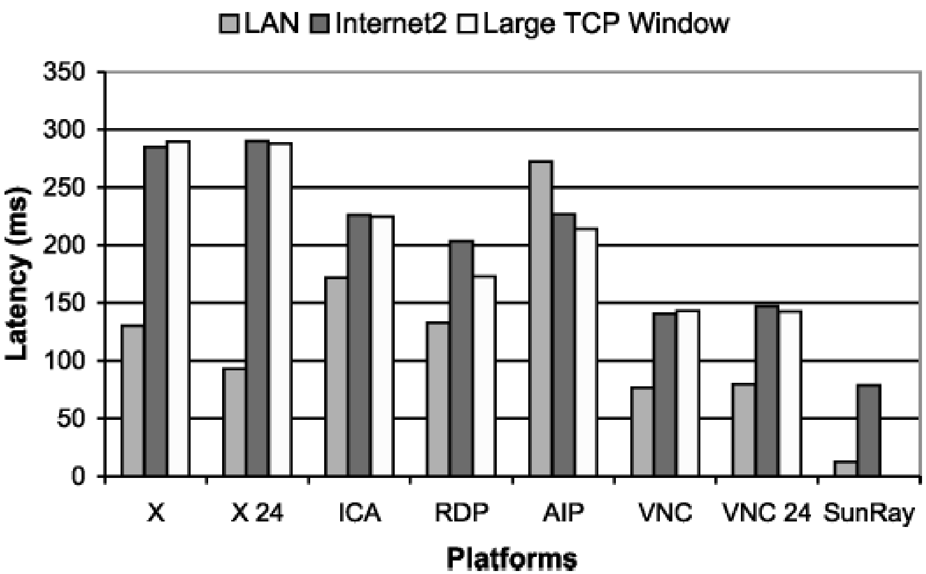
- Latency benchmark
  - Small Java applet to run 5 small microtests
    - Letter – keystroke + display a 12-point ‘A’
    - Scroll – scroll down 450 word, 58 lines, 12-poin page
    - Fill – fill screen with 320x240 red pixels
    - Red bitmap – bitmap download a 1.78KB JPG red bitmap of 320x240
    - Image – download 15.5KB JPEG image of 320x240
- Web benchmark
  - Web Text Page Load test from Ziff-Davis i-Bench + Netscape Navigator
  - Modified for slow-motion benchmarking – one page at a time
- Video playback benchmark
  - 5.11MB MPEG1 video; a 34s clip with ideal rate of 24fps
  - Modified for slow-motion benchmarking – both 1fps and 24fps rates

Latency over bandwidth

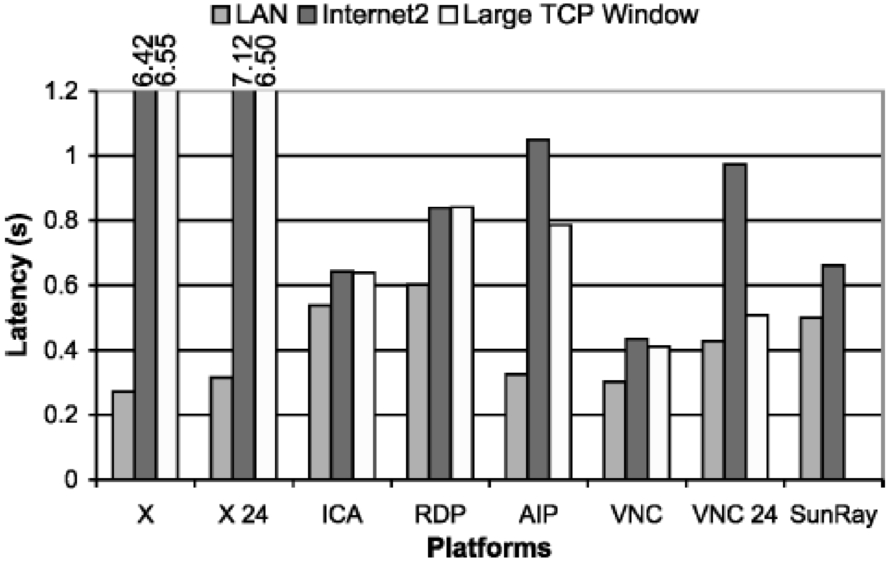


Bandwidth used – note Sun Ray is the worst

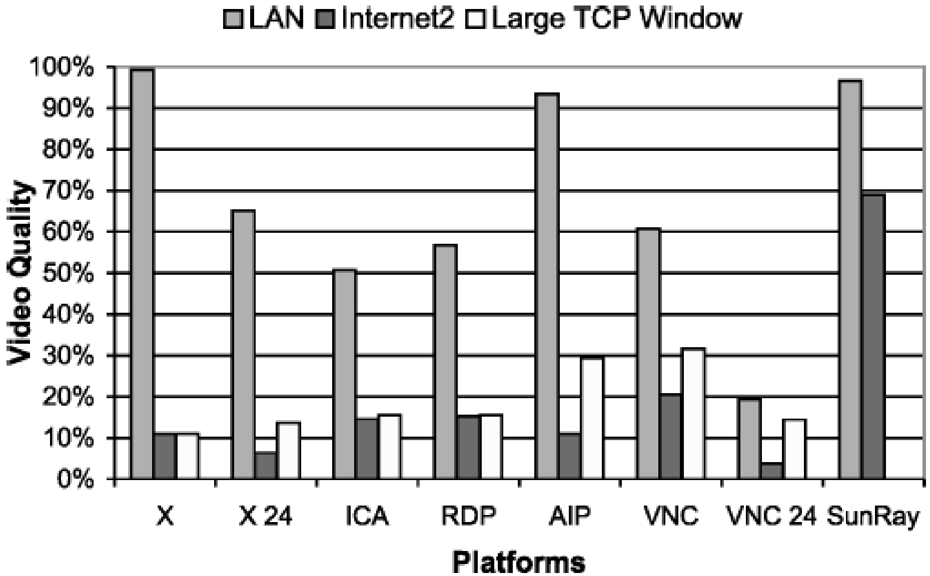
Latency for scroll (you want <50-150ms for keeping users from noticing anything – *where's Sun Ray?*



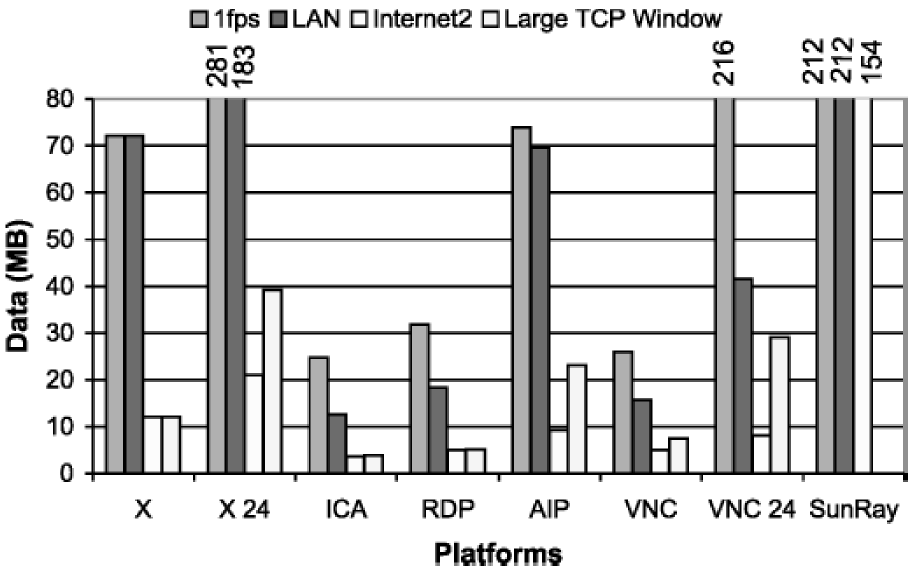
Latency over bandwidth



Web and video quality–
again note Sun Ray



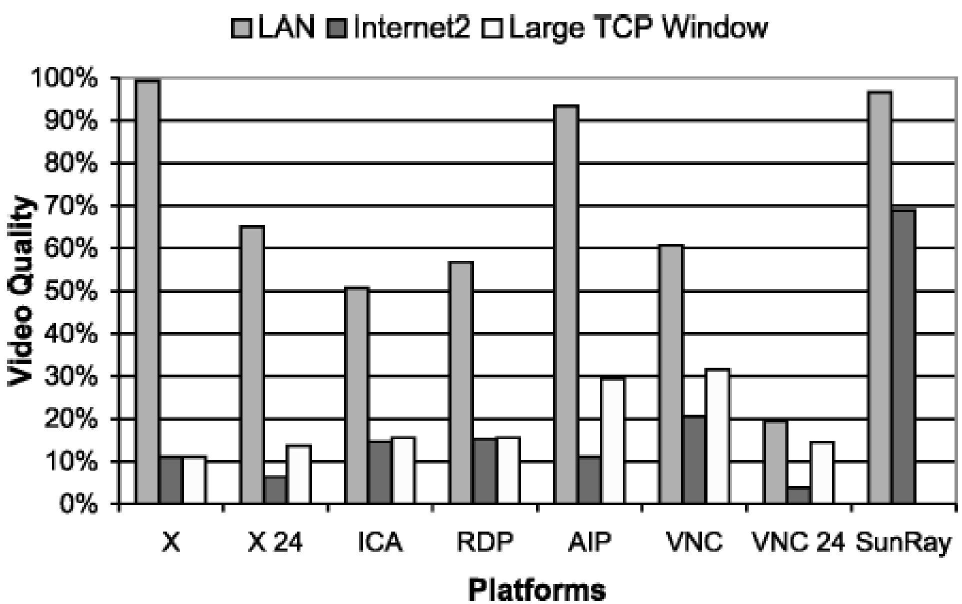
Partition client/server to minimize synch



Video data transferred – compared X LAN and WAN with Sun Ray

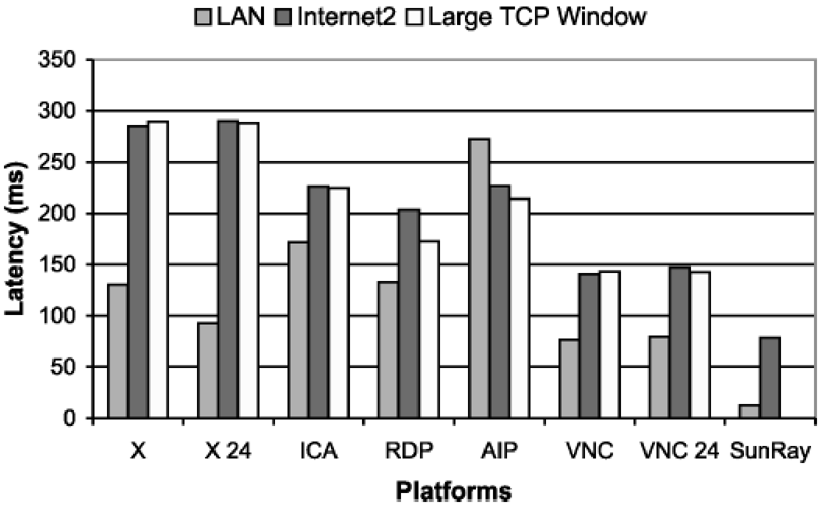
… and now look at video quality again

Most of the data is not transferred; the X display command does not complete until the client receives the frame and acks; by then the app has to skip 10 frames!



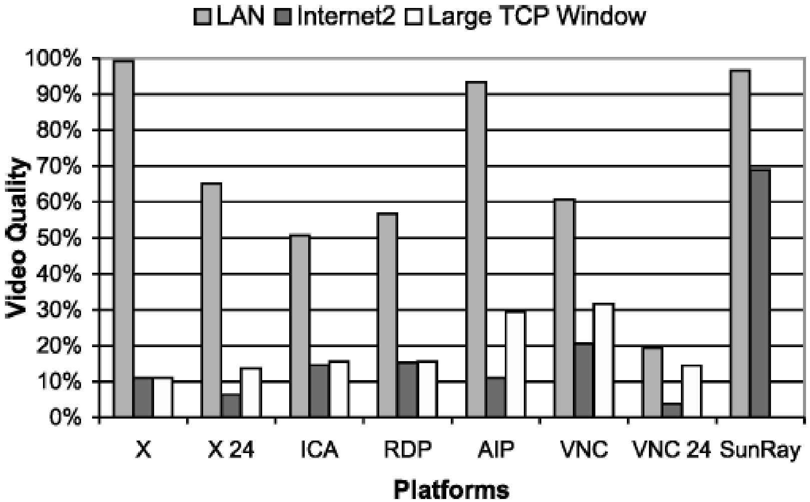
... other guidelines ...

- Use simpler, pixel-based display primitives
  - Higher-level display encodings (X) are meant to save bandwidth with text, but text is cheap
  - In general there's no difference and worst user perceived performance
  - 2d draw primitives like Sun Ray's and VNC's work best
- Use low-level forms of compression
  - "Smart", adaptive compression algorithms may give you worst than expected results



... other guidelines ...

- Adopt eager, server-push display updates
  - Both lazy and client-pull are attempts at reducing bandwidth usage – both are ideas
  - Look at lazy, client-pull VNC and eager, server pull Sun Ray



- UDP over TCP
  - Less to tune and the application knows best anyway (the thin-client layer)

Summary

- First quantitative measurement to examine the impact of WAN latency on thin-client computing
- Wide-area computing services are feasible
- Growing number of multimedia apps and available bandwidth → …
- Guidelines
  - Optimize for latency rather than bandwidth
  - Minimize need for synchronized local client window system state
  - Use simpler, pixel-based display primitives
  - Adopt eager, server-push display updates
  - Use low-level forms of compression
  - UDP over TCP