Protocols for Application and Desktop Sharing

`draft-lennox-avt-app-sharing-00`

IETF AVT Working Group
Wednesday, March 9, 2005

Jonathan Lennox/Henning Schulzrinne/Jason Nieh/Ricardo Baratto
Columbia University
{lennox,hgs,nieh,ricardo}@cs.columbia.edu

Overview: Motivation

- Want to be able to remotely view and access applications.
  - Currently: T.120, proprietary solutions, treat as video sources
- Want to share existing, unmodified applications.
  - Initial motivation: show PowerPoint slides in a SIP session.
  - **Not** doing shared application state (shared whiteboard, shared text editing).
- Want this to be integrated with the IETF session architecture.
  - Share slides as part of a SIP conference.
- Treat remote access (“vnc”, “terminal server”) and application sharing as the same problem.

Requirements Overview

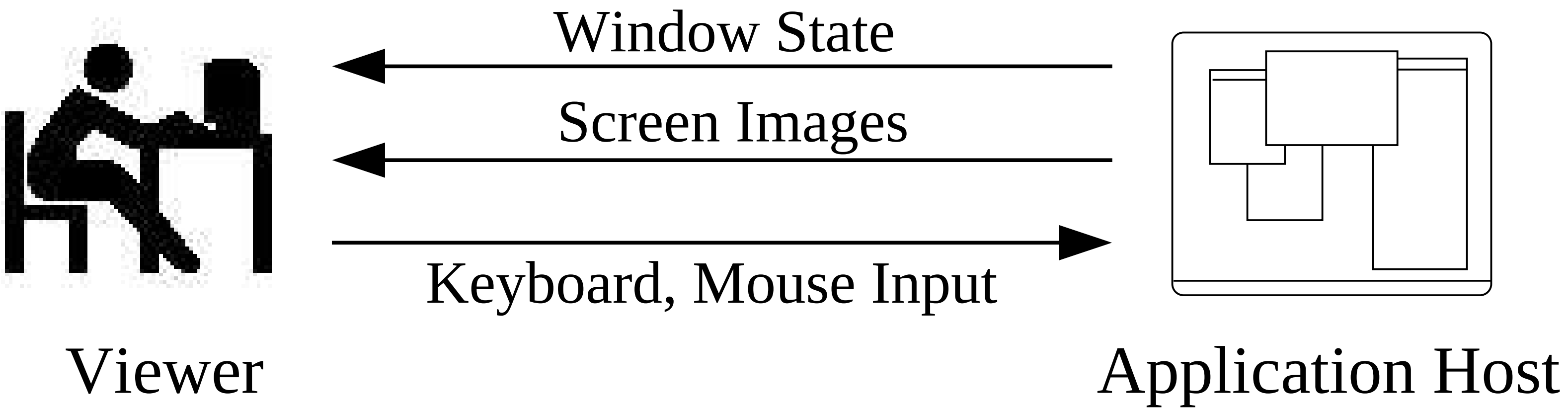
- Share both desktops (whole screens) and specific applications.
- For applications, share multiple windows, which can move around, be re-stacked, etc.
- Intelligent representation of screen images, window state, and keyboard/mouse input.
- Private, authenticated, integrity-protected, and access-controlled.
- Integrate into the IETF session architecture.
- Support diverse end systems.
- See `draft-schulzrinne-mmusic-sharing-00`.

Comparison of Approaches to Remote Application Access

Application State	Sharing-Aware
UI Elements	Special Applications (may not be sharing-aware)
Pixels and Keystrokes	**Unmodified Applications**

⇐

Components



- **Application hosts:** hosts on which applications are running; send window state and screen images to viewers.
- **Viewers:** hosts on which users access remote applications: send keyboard and mouse input to application hosts.

Protocol Components

- Window pixel data: visual contents of windows.
- Window state: create, resize, move, raise, lower, and close application windows.
- Pointer image and position: optimization, don't send the pointer as part of the pixel data.
- Keyboard and mouse input.
- Additional protocol components can be defined later; negotiate in SDP offer/answer as normal.

Transport

- Input and output protocols use RTP-over-TCP (contrans).
- Could use standard RTP-over-UDP in unusual circumstances, such as multicast. (This would probably need a reliability mechanism.)

Transport: Rationale

- Why TCP?
  - Reliability usually more important than timeliness.
  - Flow control and dynamic bandwidth adjustment crucial.
- Why RTP?
  - Natural to send data with a packetization format.
  - These packets should have timestamps, sequence numbers, variable payloads.
    - Sometimes need timing information for screen data and input (e.g. for animation, games).
  - Want to be able to use existing RTP payload formats for full-motion video.
  - No point in inventing something new.

Window Pixel Data

- “Meta-protocol” header that defines window ID, X and Y offsets.
- Encloses actual data protocol format.
- MUST support PNG images, solid-color rectangles, image copy.
- MAY support video/* MIME types.
  - Meta-protocol scheme lets existing video payload definitions be used without modifications.
  - Existing video codecs are much more efficient than “motion PNG” for actual full-motion video.
  - This may require applications to know about the sharing protocol (despite earlier requirement) to avoid multiple transcodings.

Window State

- An “application” is a stack of windows, dynamically modified.
- Windows can be created, moved, resized, raised, lowered, closed.
- Windows can have non-rectangular shapes, or be translucent. Use PNG transparency.
- Window state protocol also supports “pointer capture.”
- Window state protocol is not used in desktop sharing mode.

Pointer Representation

- Send pointer position and shape separately from window image.
- RFC 2862 (video/pointer) is defined for this, but only supports 12-bit X and Y positions.

Input Protocols

- RTP payload for mouse position and button state
  - Again, RFC 2862 handles this, but only supports 12-bit positions; also only 3 mouse buttons (no wheels).
- Keyboard state
  - Send list of keys down, locks in effect at any given time.

Open Issues: Big Picture

- Is this a useful problem to be solving?
- Is this the right architecture for a solution?
- Is AVT the right home for it?
- Do any other major pieces need to be added for an initial specification?
  - Beep.
  - Audio in general.
  - Copy and paste between viewer's remote and local apps.
  - Portholing and scaling, for small-screen devices.

Open Issues 2

- Does this need SDP extensions?
  - Some parameters can use a=fmtp: parameters (equivalent to MIME type parameters).
  - Some might better be defined as new SDP attributes.
- We'd like to send the window state protocol and the pixel images over a single TCP/RTP connection.
  - But the former should be "application", and the latter should probably be "video". Note also "image/png".
  - This isn't currently allowed.
- What's the right mechanism to secure the protocol streams?
  - TCP/RTP/SAVP? TCP/TLS/RTP/AVP?

Open Issues 3

- Should we have taskbar support?
  - Application host to viewer: window titles, list of minimized windows.
  - Viewer to application host: actions on taskbar items (unminimize, maximize, close, etc.)
  - Note that these actions on windows themselves are handled non-semantically, as mouse events on the window manager trim.

Open issues 4

- Should viewers be able to request a full screen refresh?
  - See FIR (Full Intra-Frame Request) RTCP packet, RFC 2032.
- Should RFC 2862 (video/pointer) be updated/obsoleted?
  - Screen resolution limited to 4096x4096.
  - Only three mouse buttons.

Open issues 5

- Need good names for the protocol suite as a whole, and for its various components.
  - Needed for MIME type registrations, as well as “marketing.”