

Audio/Video Transport  
Internet-Draft  
Expires: June 1, 2005

J. Lennox  
H. Schulzrinne  
J. Nieh  
R. Barrato  
Columbia U.  
December 2004

Protocols for Application and Desktop Sharing  
draft-lennox-avt-app-sharing-00

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of [section 3 of RFC 3667](#). By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she become aware will be disclosed, in accordance with [RFC 3668](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on June 1, 2005.

Copyright Notice

Copyright (C) The Internet Society (2004).

Abstract

This document defines several protocols to support accessing general graphical user interface (GUI) desktops and applications remotely, either by a single remote user or embedded into a multiparty conference. The protocols are designed to allow sharing of, and access to general windowing system applications that are not

Internet-Draft

Application and Desktop Sharing

December 2004

expressly written to be accessed remotely.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">2.</a>	Terminology . . . . .	<a href="#">4</a>
<a href="#">3.</a>	Overview . . . . .	<a href="#">5</a>
<a href="#">3.1</a>	Architecture . . . . .	<a href="#">5</a>
<a href="#">3.2</a>	Protocol Components . . . . .	<a href="#">5</a>
<a href="#">4.</a>	Common Protocol Elements . . . . .	<a href="#">6</a>
<a href="#">5.</a>	Output Protocols . . . . .	<a href="#">6</a>
<a href="#">5.1</a>	Window Identifiers and Output Meta-Format . . . . .	<a href="#">6</a>
<a href="#">5.2</a>	Window State Protocol . . . . .	<a href="#">7</a>
<a href="#">5.3</a>	Window Pixel Data . . . . .	<a href="#">9</a>
<a href="#">5.4</a>	Pointer Representation . . . . .	<a href="#">10</a>
<a href="#">6.</a>	Input Protocols . . . . .	<a href="#">10</a>
<a href="#">6.1</a>	Keyboard Input . . . . .	<a href="#">10</a>
<a href="#">6.2</a>	Pointer Position . . . . .	<a href="#">15</a>
<a href="#">7.</a>	Implementation Notes . . . . .	<a href="#">16</a>
<a href="#">8.</a>	Open issues . . . . .	<a href="#">16</a>
<a href="#">9.</a>	Security Considerations . . . . .	<a href="#">18</a>
<a href="#">10.</a>	IANA Considerations . . . . .	<a href="#">18</a>
<a href="#">11.</a>	References . . . . .	<a href="#">18</a>
<a href="#">11.1</a>	Normative References . . . . .	<a href="#">18</a>
<a href="#">11.2</a>	Informative References . . . . .	<a href="#">19</a>
	Authors' Addresses . . . . .	<a href="#">20</a>
	Intellectual Property and Copyright Statements . . . . .	<a href="#">22</a>

Internet-Draft

Application and Desktop Sharing

December 2004

## 1. Introduction

While two-party and multi-party conferencing using standards-based protocols is now common and well-developed, protocols for sharing applications are largely proprietary or based on the aging T.120 [8] suite of protocols. In this document, we define a set of protocols for application and desktop sharing.

We note that there are large similarities between remote access to an application ("remote desktop") and by multiple users sharing an application within a collaboration setting such as a multimedia call or multiparty conference. The protocols defined in this document therefore support both.

Remote access differs from video transmission of the sort for which most video encodings have been designed. In particular, screen encoding may need to be lossless and typically operates on artificial rather than natural (photographic) video input. The video input is characterized by large areas of the screen that remain unchanged for long periods of time, while others change rapidly. (However, rendering the output of a modern computer-generated animation application such as video games blurs the distinction between traditional motion video output and screen sharing.)

Unlike earlier systems, such as T.120, we believe that application sharing should be integrated into the existing IETF session model, encompassing session descriptions using the Session Description Protocol (SDP) [1] or successors and the Session Initiation Protocol (SIP) [9]. Application sharing needs many of the same control functions as other multimedia sessions, such as address binding and session feature and media negotiation. We believe that use of the session model is also beneficial for the remote desktop case, as it allows to re-use many of the well-developed session components and easily supports hybrid models, such as the delivery of desktop audio to the remote user.

Remote access to graphical applications and desktops, as defined in this document, has two important characteristics. First, the access protocol is unaware of any semantic characteristics of the applications being shared; it only transmits the visual characteristics of the windows. This is different, therefore, from shared-drawing or shared-editing tools that allow distributed modification of documents. Secondly, the protocol is designed to work with applications which were not written to be used remotely, by intercepting or simulating their connections to their native window systems. In this way, it is distinguished from systems such as the X Window System [[10](#)], which allow natively-written applications to be displayed on remote viewers.

We distinguish between local and remote users. Local users employ normal operating system mechanisms to interact with the running application. Remote users interact via the delivery protocols described here.

The application sharing problem can be divided into four components: (1) setting up a session to the node running the application, (2) transporting user input events from the remote viewers such as conference participants to the application, (3) delivering screen output from the application to the participants, (4) moderating access to shared human interface devices such as pointing devices (e.g., mice, joystick, trackball) and text input (keyboard). We refer to components (2) and (3) as the "remoting protocol". They are the focus of this document, and are described in [Section 6](#) and [Section 5](#) respectively.

Session negotiation and description can be provided by existing session setup protocols; user input access can be moderated by a floor control protocol. Thus, these two components are beyond the scope of this document, although they are important for an acceptable overall user experience.

Applications are more than just windows; they are a stack of related windows which serve the same task and are usually associated with the same process on the server. Some applications impose special constraints on the user input, e.g., through modal dialogs, which temporarily exclusively acquire input focus, and floating (always-on-top) windows.

The protocols described in this document are intended to fulfil the requirements described in the Internet-Draft Sharing and Remote Access to Applications [[11](#)].

The rest of this document is laid out as follows. [Section 2](#) defines the common terminology for normative references. [Section 3](#) gives an overview of the protocol's architecture and components. [Section 4](#) defines common elements of the output and input protocols, which are then further described in [Section 5](#) and [Section 6](#) respectively. [Section 7](#) gives implementation notes, and [Section 8](#) discusses open issues with the design of the protocol. Finally, [Section 9](#) discusses security considerations, and [Section 10](#) gives IANA considerations.

## [2](#). Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [RFC 2119](#) [[2](#)] and indicate requirement levels for compliant implementations.

## [3](#). Overview

### [3.1](#) Architecture

Application and desktop sharing consists of two classes of components: "viewers" and "application hosts". Viewers receive remote graphics and provide input. Application hosts receive input from local and remote users, and host and transmit applications and graphics.

The application and desktop sharing models defined in this document are integrated into the IETF conferencing model. In particular, the Session Initiation Protocol (SIP) [[9](#)] is used to initiate and control remote access. This allows the use of existing SIP mechanisms for confidentiality, authentication and authorization, user location, conferencing, etc.

In the IETF conferencing model media sessions can consist of multiple participants; this document's protocols are designed to work in this case. The various Centralized Conferencing (XCON) [[12](#)] control protocols can be used for floor control, to determine which member of

a conference is permitted to send input to an application host at any given time. Conferencing also gives rise to issues of late-joiners; the protocol is designed to make it relatively easy for a protocol relay, which receives input from one application host and forwards it to multiple viewers, to send all the necessary information about a sharing session to new conference arrivals. Similarly, it is possible to record and replay a shared application session without semantic awareness of the protocol.

### [3.2](#) Protocol Components

The three core components of desktop sharing are: input protocols which represent user input from keyboards or pointing devices such as mice, trackballs, or touchscreens; an output protocol which can represent screen pixels and related data; and a negotiation mechanism which can convey attributes of the session such as the desired size of the screen. In addition, application sharing requires a mechanism to represent window state, position and stacking. In this document, the negotiation mechanism is defined in [Section 4](#); output protocols, including window-state handling, are defined in [Section 5](#); and input protocols are defined in [Section 6](#).

Additional, optional mechanisms can enhance both window and application sharing. Additional input mechanisms such as joysticks or other game controllers can be supported; audio streams can be associated with a desktop or application; viewer-side scaling and porthole requests can be used to optimize transmission of data to

viewers with a small screen; and it is often useful to allow copy-and-paste between applications running on a viewer and those running on an application host. This document does not define any such extensions; they may be defined elsewhere.

## [4.](#) Common Protocol Elements

Protocol negotiation is carried out using the Session Description Protocol (SDP) [[1](#)], while all input and output protocols run over the Real-Time Protocol (RTP) [[3](#)]. In most use cases for application and desktop sharing, reliability is more important than latency, and flow control and dynamic bandwidth adjustment are crucial. As such, viewers and application hosts SHOULD use RTP Framing [[4](#)] to send the RTP packets over TCP, unless there is a strong reason, such as the

need to distribute a desktop session over multicast, to do otherwise.

## 5. Output Protocols

### 5.1 Window Identifiers and Output Meta-Format

A shared application consists of a set of overlapping windows, usually rectangular. Each window needs a unique identifier, and most output data (other than audio or other non-visual output mechanisms not specified here) needs to be associated with a particular window.

Windows in an application need to be created and destroyed relatively frequently. Re-negotiating SDP descriptions whenever a window is opened or closed would therefore not be practical, and so data for multiple windows needs to be multiplexed into a single RTP stream. Since multiple windows are from a single source, multiplexing on the RTP SSRC (synchronization source) field would not be appropriate. Instead, each window is assigned a unique window identifier.

Rather than require each payload type to define a field to carry this window identifier we instead define a payload meta-format which precedes each payload. The meta-format carries the relevant window identifier and coordinates, and is then followed by the actual data for the particular payload being sent. (This allows existing payload type definitions to be re-used in the context of a window.)

This protocol has three mandatory format-specific parameters, which are carried in an SDP "a=fmtp:" parameter. The parameter "height" indicates the desktop height in pixels; "width" indicates the desktop width in pixels. All images and other coordinates sent for this protocol must lie within these boundaries. The third parameter is "mode", which can take the value "desktop", indicating one big drawing pane, or "application", indicating that individual windows will be created and destroyed as needed, and all drawing will occur in

individual windows.

In application mode, SDP "i=" lines for this protocol SHOULD contain a human-readable description of the application.

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			





creation, destruction, resizing, raising and lowering, positioning, and characteristics of application windows when the protocol is being run in application mode. It MUST NOT be used in desktop mode.

The specific details and packet format of this protocol are not yet defined. The rest of this section describes it at a high level.

Creating, moving, resizing, and raising or lowering a window are indicated by the same message. This message starts with the common meta-protocol header, whose coordinates indicate the position of the window relative to the desktop. Following this is a code indicating that this is a "window state" message, and the window's X and Y sizes as 32-bit integers. Finally, the message contains a list of all the application's windows, in Z order, bottom to top. (Listing the entire Z order in every message helps prevent the Z order list from getting out of sync between viewers and the application host.) A flag in the Z order list can indicate that some windows should be considered "always on top", and float above all non-"on top" other windows on the viewer (shared or not).

The size and position given for a window includes all the "trim" provided by its window manager -- title bars, frames, and the like. (Otherwise, the remoting protocol would need to include input and output messages to indicate window state changes and manipulation.)

Non-rectangular and translucent windows can additionally have an alpha channel specified. This is sent as a black-and-white or grayscale PNG [\[5\]](#) image corresponding to the window's transparent or translucent pixels. Window alpha channels can change dynamically.

Window removal consists of the meta-protocol header followed by a "window remove" code. On receipt of this message, a viewer erases the corresponding window and removes it from the Z order list. The X and Y coordinates given in the meta-protocol are ignored and SHOULD be zero.

An additional message indicates the "pointer capture", in which a window indicates that it should exclusively receive all pointer events until it indicates otherwise. This is necessary when menus are pulled down, for example; a window with a pulled-down menu receives a "release menu" mouse click whether or not the cursor is still over the original window. "Stop pointer capture" is the same message, with a flag set.

The window state protocol may need to carry additional information, as well; see the open issues list in [Section 8](#).

Internet-Draft

Application and Desktop Sharing

December 2004

### [5.3](#) Window Pixel Data

There are three basic window data operations: pixel images, fills, and block copies. Each operation uses the common meta-protocol header. Each operation has its own MIME type, and thus a unique RTP payload type in the meta-protocol PT field.

Pixel images contain arbitrary graphical data to be applied to windows. They are conveyed as PNG [\[5\]](#) images. The PNG image follows the meta-protocol header (which indicates the offset of the image within the window or desktop) and consists of an area of the screen to be updated. If the PNG image contains an alpha channel, the image is composited with the existing contents of the window or desktop. (The PNG images defining the initial graphical contents of a window or desktop MUST NOT contain alpha channels.) In window mode, if a window has an alpha channel with completely transparent pixels -- i.e., if a window is non-rectangular -- the corresponding pixels in the PNG image are ignored.

As an optimization, two additional window data operations are defined. A fill defines an area of a window to be filled by a single solid color. Following the meta-protocol header, it consists of a height and width (specified as 32-bit coordinates), followed by the fill color. Colors are specified as one byte each of Red, Green, and Blue, i.e. as PNG color type 2 with 8-bit sample depth. (Color sample depths greater than 8 bits per channel cannot be specified with the fill operation, and must use the general PNG pixel image form.)

The block copy operation copies a region of a desktop or window from one position to another. Following the meta-protocol header (which indicates the destination position) are the source position and size (both as 32-bit coordinates). The destination region MAY overlap the source region. Both the source and destination regions MUST NOT extend beyond the boundaries of the window (in window mode) or desktop (in desktop mode). In window mode, cross-window moves are not supported. Portions of the source region which do not overlap with the destination region remain unmodified.

Additionally, if the viewer and application host negotiate support for other video/\* MIME types, video streams can be sent following the meta-protocol header. For video this will often be more efficient than sending raw screen images.

In window mode, graphics to be drawn MUST NOT extend beyond the boundaries of the window; in either mode, images to be drawn MUST NOT extend beyond the defined borders of the desktop. (Window-related images such as drop-down menus or tooltips which can extend beyond

the boundaries of a window SHOULD be transmitted as separate windows.)

#### [5.4](#) Pointer Representation

For efficiency, pointers can be represented separately from other window data. This is accomplished by transmitting, in a special protocol, PNG [\[5\]](#)s with alpha channels and hotspots for the pointers' images, and then [RFC 2862](#) [\[6\]](#) streams to indicate pointers' positions. These protocols still need to be defined in detail.

### [6.](#) Input Protocols

#### [6.1](#) Keyboard Input

The viewer represents keyboard input to the server by sending a list of depressed keys, updated whenever this state changes. (Note that this is unlike how keys are represented in most window systems, which instead use individual key-down and key-up events. The latter can be derived from the former.) Key repetition is handled by the application host.

There are two types of keys that can be represented. "Encoding keys" are keys that encode a specific Unicode [\[7\]](#) character, whereas "virtual keys" do not. Encoding keys are indicated by the Unicode value of the character they encode.

Virtual keys are indicated by codes from the tables of virtual keycodes listed in Figure 2, Figure 3, Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, and Figure 9. These codes are those of the X Window System [\[10\]](#), taken from the header file <X11/keysymdef.h>. They are all of X's keycodes with 0xFFnn values, except for the XK\_KP\_\* keypad keys.

Unicode control characters -- characters in the range 0x0000 - 0x001f or 0x007f - 0x009f -- MUST NOT be sent. Instead, the corresponding

virtual key, or the virtual key Control (Left) or Control (Right) (0xE3 or 0xE4) plus a Unicode codepoint, should be used.

Name	Code	Note
----	----	----
Backspace	0x08	Back space, back char
Tab	0x09	
Linefeed	0x0A	Linefeed, LF
Clear	0x0B	
Return	0x0D	Return, enter
Pause	0x13	Pause, hold
Scroll Lock	0x14	
Sys Req	0x15	
Escape	0x1B	
Delete	0xFF	Delete, rubout

These codes have been chosen to map to ASCII, for convenience of programming, but could have been arbitrary (at the cost of lookup tables in viewer code).

Figure 2: Virtual keycodes: teletype keys

Name	Code	Note
----	----	----
Multi key	0x20	Multi-key character compose
Code Input	0x37	
Single Candidate	0x3C	
Multiple Candidate	0x3D	
Previous Candidate	0x3E	

Figure 3: Virtual keycodes: international and multi-key character

Name	Code	Note
----	----	----
Kanji	0x21	Kanji, Kanji convert
Muhenkan	0x22	Cancel Conversion
Henkan Mode	0x23	Start/Stop Conversion
Romaji	0x24	to Romaji
Hiragana	0x25	to Hiragana
Katakana	0x26	to Katakana
Hiragana/Katakana	0x27	Hiragana/Katakana toggle
Zenkaku	0x28	to Zenkaku
Hankaku	0x29	to Hankaku
Zenkaku/Hankaku	0x2A	Zenkaku/Hankaku toggle
Touroku	0x2B	Add to Dictionary
Massyo	0x2C	Delete from Dictionary
Kana Lock	0x2D	Kana Lock
Kana Shift	0x2E	Kana Shift
Eisu Shift	0x2F	Alphanumeric Shift
Eisu Toggle	0x30	Alphanumeric toggle
Kanji Bangou	0x37	Codeinput
Zen Koho	0x3D	Multiple/All Candidate(s)
Mae Koho	0x3E	Previous Candidate

Note that some of these codes are also used for equivalent Hangul keyboard keys listed in Figure 9.

Figure 4: Virtual keys: Japanese keyboard support

Name	Code	Note
----	----	----
Home	0x50	
Left	0x51	Move left, left arrow
Up	0x52	Move up, up arrow
Right	0x53	Move right, right arrow
Down	0x54	Move down, down arrow
Prior	0x55	Prior, previous
Page Up	0x55	
Next	0x56	Next
Page Down	0x56	
End	0x57	EOL
Begin	0x58	BOL

Figure 5: Virtual keycodes: cursor control and motion

Name	Code	Note
----	----	----
Select	0x60	Select, mark
Print	0x61	
Execute	0x62	Execute, run, do
Insert	0x63	Insert, insert here
Undo	0x65	Undo, oops
Redo	0x66	Redo, again
Menu	0x67	
Find	0x68	Find, search
Cancel	0x69	Cancel, stop, abort, exit
Help	0x6A	Help
Break	0x6B	
Mode switch	0x7E	Character set switch (*)
Num Lock	0x7F	

(\*) The "Mode switch" key is variously used on Katakana, Arabic, Greek, Hebrew, and Hangul keyboards to switch between the Roman and native alphabets.

Figure 6: Virtual keycodes: miscellaneous functions

Name	Code
----	----
F1	0xBE
F2	0xBF
F3	0xC0
F4	0xC1
F5	0xC2
F6	0xC3
F7	0xC4
F8	0xC5
F9	0xC6
F10	0xC7
F11/L1	0xC8
F12/L2	0xC9
F13/L3	0xCA
F14/L4	0xCB
F15/L5	0xCC
F16/L6	0xCD
F17/L7	0xCE
F18/L8	0xCF
F19/L9	0xD0
F20/L10	0xD1
F21/R1	0xD2
F22/R2	0xD3
F23/R3	0xD4

F24/R4	0xD5
F25/R5	0xD6
F26/R6	0xD7
F27/R7	0xD8
F28/R8	0xD9
F29/R9	0xDA
F30/R10	0xDB
F31/R11	0xDC

F32/R12	0xDD
F33/R13	0xDE
F34/R14	0xDF
F35/R15	0xE0

Sun keyboards and a few other manufacturers have additional Left and Right function key groups on the left and/or right sides of the keyboard.

Figure 7: Virtual keycodes: auxiliary functions

Name	Code	Note
----	----	----
Shift (Left)	0xE1	
Shift (Right)	0xE2	
Control (Left)	0xE3	
Control (Right)	0xE4	
Caps Lock	0xE5	
Shift Lock	0xE6	
Meta (Left)	0xE7	Windows (Microsoft); Option (Macintosh)
Meta (Right)	0xE8	Windows (Microsoft); Option (Macintosh)
Alt (Left)	0xE9	Command (Macintosh)
Alt (Right)	0xEA	Command (Macintosh)
Super (Left)	0xEB	
Super (Right)	0xEC	
Hyper (Left)	0xED	
Hyper (Right)	0xEE	

Application hosts which lack right-hand versions of modifiers SHOULD treat them as though the left-hand version had been received.

Figure 8: Virtual keys: modifiers



----	----	----
Hangul	0x31	Hangul start/stop(toggle)
Hangul Start	0x32	Hangul start
Hangul End	0x33	Hangul end, English start
Hangul Hanja	0x34	Start Hangul->Hanja Conversion
Hangul Jamo	0x35	Hangul Jamo mode
Hangul Romaja	0x36	Hangul Romaja mode
Hangul Code Input	0x37	Hangul code input mode
Hangul Jeonja	0x38	Jeonja mode
Hangul Banja	0x39	Banja mode
Hangul Pre-Hanja	0x3A	Pre Hanja conversion
Hangul Post-Hanja	0x3B	Post Hanja conversion
Hangul Single Candidate	0x3C	Single candidate
Hangul Multiple Candidate	0x3D	Multiple candidate
Hangul Previous Candidate	0x3E	Previous candidate
Hangul Special	0x3F	Special symbols

Figure 9: Virtual keys: Hangul (Korean)

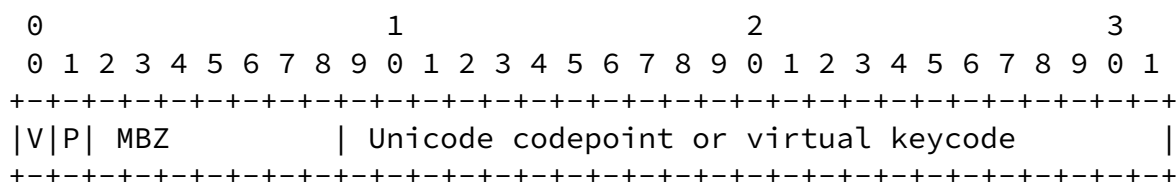


Figure 10: One entry in the keyboard state list

Figure 10 illustrates the format of entries in the keyboard state list. If the "V" bit is set, it indicates that the codepoint number indicates a virtual keycode, otherwise it indicates a Unicode codepoint. If the "P" bit is set, it indicates that it is the version of a key from a separate keypad. If the server does not have a concept of a separate keypad, or the key indicated does not appear on a keypad, the "P" bit MAY be ignored.

The highest Unicode codepoint is 0x10ffff; thus, all Unicode characters fit comfortably in the 24-bit codepoint field of the keycode format. For non-virtual, non-keypad keys, for which the V and P bits are both zero, the format is identical to a UTF-32BE encoding of the same character. (The format can, thus, be considered alternatively as UTF-32BE bitwise-or'd with 0x80000000 for "V" and 0x40000000 for "P".)

## 6.2 Pointer Position

The viewer indicates pointer position to the application host using

the media type video/pointer, defined in [RFC 2862](#), the RTP Payload Format for Real-Time Pointers [6].

## [7.](#) Implementation Notes

Application hosts shouldn't blindly send every screen update they receive down the RTP channel. Instead, they should monitor the state of their TCP transmission buffers (through mechanisms such as the `select()` command) and only send the most recent screen data when there is not a backlog. This will prevent screen latency for rapidly-changing images, when a viewer usually only needs to see the final state of the image.

To conserve bandwidth, application hosts SHOULD use PNG's palette or grayscale image format wherever possible, and SHOULD use a minimal palette and image bit depth, subject to encoding delay constraints. In particular, two-color images, and one-color images drawn over the existing image, SHOULD use a one-bit PNG with a two-entry palette, in the latter case with a transparency chunk.

In window mode, application hosts SHOULD be aware of unshared local windows on the host. If an unshared window obscures a shared window, the application host SHOULD obscure its contents (through a mechanism such as transmitting a neutral color) so that the viewer experience reflects as closely as possible the experience on the host. Application hosts MAY choose to treat portions of windows obscured by other shared windows the same way.

## [8.](#) Open issues

We need to determine what mechanism to recommend to secure the input and output streams. The two logical possibilities would be Secure RTP [13] and Transport-Layer Security (TLS) [14]. Either would work; neither is currently specified for connection-oriented RTP (neither TCP/RTP/SAVP nor TCP/TLS/RTP/AVP is defined). One or the other ought to be recommended to facilitate interoperability.

It seems likely that "beep" needs to be defined specially, as an output type, and needs to be defined separately from other audio channels. Many systems allow beeps to be rendered visually, either for accessibility for the deaf or because systems are being used in quiet environments.

We need a name for the meta-protocol of [Section 5.1](#). It's also unclear whether it should have an `application/*` or `video/*` MIME type.

SDP doesn't normally allow you to send traffic with different top-level MIME types over the same RTP channel. Do we need to add an

extension to work around this? Some of the RTP payloads described in this document should pretty clearly be application/\*, some should be video/\*, some (beeping) audio/\*, PNG is already defined as image/png, etc.

All the payload types need MIME-type assignments and RTP payload characteristics (sample rate, use of marker bit, etc.) defined.

Is there anything useful that can be done with the MBZ bits of the meta-protocol?

What other information needs to be carried in the window state protocol? One particular concern is taskbar support. Window information that might be carried to support taskbars includes the window title, a list of minimized windows, and whether each window should be listed in the taskbar. Taskbar support also requires an input protocol to support taskbar actions (right-clicking on a taskbar item): unminimize, maximize, close, etc.

Another flag that might be carried by the window state protocol is whether a window is "taggable", i.e. whether it is a good candidate for a viewer-side marker indicating that this is a shared application. Top-level windows would typically get this, while subsidiary windows such as dialog boxes would not. (This is a feature of T.128.)

Do we need to define what an un-drawn-upon window looks like? T.128 seems to assume that windows are transparent until drawn to, and uses this fact: a server can define a full-screen window on top of all others, and draw directly to it. I don't think this is necessary, but it's an important consideration.

Should pointer images be window-associated? Should there be a pointer image cache? (T.128 has one; VNC doesn't.)

[RFC 2862](#) only supports 12-bit positioning for the mouse pointer. I believe this is already too small; screens wider than 4096 pixels already exist, especially virtual desktops. Additionally, we want to support additional mouse information, most notably mousewheels. A

protocol obsoleting [RFC 2862](#) is probably in order.

Do we want to support a mechanism by which viewers can request a full screen refresh, analogous with [RFC 2032](#) [15]'s Full Intra-frame Request (FIR) RTCP packet?

One other optimized pixel transmission operation that could be used is the tiling operation: transmit one image and a number of times it should be repeated horizontally and vertically. Would this be worth

the bandwidth/complexity tradeoff? (Note that it can be emulated without too much overhead by the copy operation.)

Is it necessary for viewers and application hosts to be able to negotiate the maximum supported size and color depth of pointers? This would presumably be a format parameter on the pointer image representation payload type.

Do we want to support multiple payloads in one RTP packet, either reusing or inspired by [RFC 2198](#) [16]?

How should "lock" key state (caps lock, num lock, scroll lock) be represented in the keyboard input protocol? Should there be flags of some sort? Should there be separate virtual keycodes for "lock key depressed" and "lock state enabled"? Should this simply be handled on the viewer side, altering the keycodes that are sent? (This works for caps and num lock, not so well for scroll lock.)

## [9.](#) Security Considerations

Both input and output data may be highly sensitive. For example, input data may contain user passwords. Thus, encryption of all user input is likely to be required. For some applications, such as sharing slides during a public lecture, confidentiality for user output may not be required. Given the broad set of applications, viewers and application hosts MUST support or be able to leverage end-to-end confidentiality and integrity protection mechanism.

Application sharing inherently exposes the shared applications to risks by malicious participants. They may, for example, access resources beyond the application itself, e.g., by installing or running scripts. It may be difficult to constrain access to specific

user data, e.g., a specific set of slides, unless the user application can be sandboxed or run in some kind of "jail", with the sandbox control outside the view of the remoting protocol.

## 10. IANA Considerations

TODO; MIME type definitions for everything.

## 11. References

### 11.1 Normative References

- [1] Handley, M., Jacobson, V. and C. Perkins, "SDP: Session Description Protocol", [draft-ietf-mmusic-sdp-new-21](#) (work in progress), October 2004.

Lennox, et al.

Expires June 1, 2005

[Page 18]

---

Internet-Draft

Application and Desktop Sharing

December 2004

- [2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [3] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, [RFC 3550](#), July 2003.
- [4] Lazzaro, J., "Framing RTP and RTCP Packets over Connection-Oriented Transport", [draft-ietf-avt-rtp-framing-contrans-03](#) (work in progress), July 2004.
- [5] Duce, D., "Portable Network Graphics (PNG) Specification (Second Edition)", W3C REC REC-PNG-20031110, November 2003.
- [6] Civanlar, M. and G. Cash, "RTP Payload Format for Real-Time Pointers", [RFC 2862](#), June 2000.
- [7] International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS)", ISO Standard 10646, December 2003.

### 11.2 Informative References

- [8] International Telecommunication Union, "Data Protocols for

Multimedia Conferencing", ITU-T Recommendation T.120, July 1996.

- [9] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", [RFC 3261](#), June 2002.
- [10] Scheifler, R., "X Window System Protocol", X Consortium Standard X Version 11, Release 6.7, November 2004.
- [11] Schulzrinne, H., "Sharing and Remote Access to Applications", [draft-schulzrinne-mmusic-sharing-00](#) (work in progress), October 2004.
- [12] Barnes, M. and C. Boulton, "A Framework for Centralized Conferencing", [draft-barnes-xcon-framework-00](#) (work in progress), October 2004.
- [13] Baugher, M., "The Secure Real-time Transport Protocol", [draft-ietf-avt-srtp-09](#) (work in progress), July 2003.
- [14] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session

Lennox, et al.

Expires June 1, 2005

[Page 19]

---

Internet-Draft

Application and Desktop Sharing

December 2004

Description Protocol (SDP)", [draft-ietf-mmusic-comedia-tls-02](#) (work in progress), October 2004.

- [15] Turletti, T., "RTP Payload Format for H.261 Video Streams", [RFC 2032](#), October 1996.
- [16] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J., Vega-Garcia, A. and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", [RFC 2198](#), September 1997.
- [17] International Telecommunication Union, "Multipoint Application Sharing", ITU-T Recommendation T.128, February 1998.

#### Authors' Addresses

Jonathan Lennox  
Columbia University Department of Computer Science

450 Computer Science  
1214 Amsterdam Ave., M.C. 0401  
New York, NY 10027  
US

Phone: +1 212 939 7018  
EMail: lennox@cs.columbia.edu

Henning Schulzrinne  
Columbia University Department of Computer Science  
450 Computer Science  
1214 Amsterdam Ave., M.C. 0401  
New York, NY 10027  
US

Phone: +1 212 939 7004  
EMail: hgs+mmusic@cs.columbia.edu

Jason Nieh  
Columbia University Department of Computer Science  
450 Computer Science  
1214 Amsterdam Ave., M.C. 0401  
New York, NY 10027  
US

Phone: +1 212 939 7000  
EMail: nieh@cs.columbia.edu

Ricardo Baratto  
Columbia University Department of Computer Science  
450 Computer Science  
1214 Amsterdam Ave., M.C. 0401  
New York, NY 10027  
US

Phone: +1 212 939 7000  
EMail: ricardo@cs.columbia.edu

#### Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to



pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at [ietf-ipr@ietf.org](mailto:ietf-ipr@ietf.org).

#### Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

#### Copyright Statement

Copyright (C) The Internet Society (2004). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

#### Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.