Screen View Protocol

Josh Brown

November 24, 2021

Contents

1 Abstract

Screen View is an end to end encrypted remote screen viewing and controlling software. This document describes the protocols necessary to make it function.

2 Definitions

The following definitions are used:

- Host - The user that wants to share their screen to the Client
- Client - The user that wants to view and control the Host's screen
- Server - The intermediary server used for routing and proxying data between
- Display - A rectangular visual region which may or may not be controllable.
- Controllable - A *Display* that accepts keyboard and mouse input.

3 Remote Visual Display (RVD) Protocol

The RVD protocol is used to communicate mouse input, keyboard input, frame data, and clipboard data from the Host to the Client.

All messages can occur over either TCP or UDP but it is strongly recommended that the noted transport protocol is used.

With the exception of the *Handshake* messages. All RVD messages first byte contain a number to indicate the message type.

3.1 Handshake - TCP

3.1.1 ProtocolVersion

Handshaking begins by the Host sending the client a *ProtocolVersion* message. This lets the Client know the verison supported by the Host.

The *ProtocolVersion* message consists of 12 bytes interpreted as a string of ASCII characters in the format "RVD xxx.yyy" where xxx and yyy are the major and minor version numbers, padded with zeros.

Host →Client

version (11 bytes) - "`RVD 001.000`"

The client replies back either `0` to indicate the version is not acceptable and that the handshake has failed or `1` if the version is acceptable to the client and the handshake as succeeded. If 0 is sent, all communication should cease and and error should be displayed to user.

Client →Host

ok (1 byte) - `00000000` or `00000001`

3.1.2 Initialization

Once the handshake has succeeded the Host responds with a *DisplayChange* message.

3.2 Control messages - TCP

Control messages are messages that instruct client about changes regarding the Host.

3.2.1 DisplayChange

A *DisplayChange* message informs the client about the available *Display*s. RVD supports up to 255 displays.

Host →Client

type (1 byte) - 1
clipboard-readable (1 byte) - 0 or 1
number-of-displays (1 byte) - 1-255
displays-information (variable bytes)
DisplayInformation *number-of-displays* times

Each *Display* has an associated *DisplayInformation*. *displays-information* contains *number-of-displays DisplayInformation*'s. A *DisplayInformation* is defined below:

display-id (1 byte)
width (2 bytes) - number of pixels of the width of this display
height (2 bytes) - number of pixels of the width of this display
cell-width (2 bytes) - number of pixels of the width of a cell in the grid
cell-height (2 bytes) - number of pixels of the height of a cell in the grid
access (1 byte) - defined below
name-length (1 byte) - length of the name
name (*name-length* bytes) - display name (UTF-8)

Restrictions:

- *cell-width* must be less than *width*. *cell-height* must be less than *height*.

- The *access* byte defines what type of access is available for the display. The bits of the *access* byte are described below in Big Endian.

Bit	Description
0	Flush
1	*Controllable*
2 3 4 5 6 7	Reserved for future use

If the *Controllable* bit is 1 and the *clipboard-readable* byte is set to 1, then the clipboard is writable. The *Controllable* bit should be consistent throughout all displays.

The *Flush* bit indicates whether this display has changed, specifically if this *display-id* refers to a different *Display* than the same *display-id* did in the previous *DisplayChange* message. In initialization, this should always be 1 (as there is no previous *DisplayChange*). If the display hasn't changed (0) then the frame data may be maintained. If *Flush* is 0, *width*, *height* must remain the same as the previous *DisplayChange* specified for the *display-id*.

3.2.2 DisplayChangeReceived - TCP

The *DisplayChangeReceived* message is sent in reply after receiving a *DisplayChange* message. It indicates to the Host they may start sending *FrameData* referencing the new *DisplayInformation* in the most recent *DisplayChange*.

Client →Host

type (1 byte) - 2

3.2.3 MouseLocation - TCP/UDP

The *MouseLocation* message send information about where the mouse is currently on the screen. The Host sends this information periodically throughout the session. The Host should send a *MouseLocation* update when mouse input is received from the Host's system or in reply when it receives a *MouseInput*.

Host →Client

type (1 byte) - 3
display-id (1 byte) - 0-255
x-location (2 bytes) - x coordinate of the mouse
y-location (2 bytes) - y coordinate of the mouse

3.3 Input - TCP/UDP

Input messages (including *MouseLocation*) can be sent over TCP or UDP. TCP is preferred in most situations. However, in situations where speed is prioritized over the guarantees TCP provides (such as gaming), UDP can be used.

3.3.1 MouseInput

Client →Host

type (1 byte) - 4
display-id (1 byte) - 0-255
x-position (2 bytes) - x coordinate of the mouse
y-position (2 bytes) - y coordinate of the mouse
button-mask (1 byte) - described below

Indicates either pointer movement or a pointer button press or release. The pointer is now at (x-position, y-position), and the current state of buttons 1 to 8 are represented by bits 0 to 7 of button-mask respectively, 0 meaning up, 1 meaning down (pressed).

On a conventional mouse, buttons 1, 2 and 3 correspond to the left, middle and right buttons on the mouse. On a wheel mouse, each step of the wheel is represented by a press and release of a certain button. Button 4 means up, button 5 means down, button 6 means left and button 7 means right.

3.3.2 KeyInput

The *KeyInput* event sends key presses or releases.

Client →Host

type (1 byte) - 5
down-flag (1 byte) - 0 or 1 to indicate whether the key is now pressed or released
key (4 bytes) - "keysym"

Details can be found at the RFB Spec

3.4 Clipboard - TCP

3.4.1 ClipboardTypeRequest

Used to request clipboard types the Host supports.

Client →Host

type (1 byte) - 6

3.4.2 ClipboardTypeResponse

Response to the *ClipboardTypeRequest*

Host →Client

type (1 byte) - 7
number-of-clipboard-types (1 byte) - 0-255
data (variable bytes) - described below

number-of-clipboard-types is always 0 if *clipboard-readable* is 0.

data contains *number-of-clipboard-types ClipboardType*s. *ClipboardType* is defined below.

type-length (1 byte) - 1-255
type-name(*type-length* bytes) - type name in ASCII

3.4.3 CopyRequest

This is a request for a keyboard contents. It can be made by either the Client or the Host.

Client $\leftrightarrow$ *Host*

type (1 byte) - 8
type-length (1 byte) - 0-255
type-name(*type-length* bytes) - type name in ASCII

3.4.4 CopyResponse

CopyResponse message is a response to a *CopyRequest*.

Client $\leftrightarrow$ *Host*

type (1 byte) - 9
accepted (1 byte) - 0 or 1
type-length (1 byte) - 0-255
type-name(*type-length* bytes) - type name in ASCII
content-length (4 bytes) - the length of the content (maximum 2^{24} bytes or 16MB)
data (content-length bytes)

accepted indicates whether the *CopyRequest* was accepted. If 0, the rest of the message is empty. If *clipboard-readable* is 0, *accepted* is always 0. A Client or Host many send this message without a request. If a *CopyResponse* is unsolicited, then *accepted* must be 1.

data is zlib compressed.

3.4.5 A note on Pasting

There is a no paste message. To paste data an unsolicited *CopyResponse* may be sent and then the keyboard shortcut (ctrl+v or cmd+v) should be sent via the *KeyboardMessage*

3.5 FrameData - UDP

The *FrameData* message contains an update of a particular cell on a particular *Display*.

type (1 byte) - 6
sequence-number (4 bytes)
cell-number (2 bytes)
size (2 bytes)
data (*size* bytes)

sequence-number is an incrementing 32-bit counter for each *FrameData* sent *data* contains jpeg pixel data of the updated cell.