Excerpt from RFC 4254, "The Secure Shell (SSH) Connection protocol", by T. Ylonen and C. Lonvick.

5. Channel Mechanism

All terminal sessions, forwarded connections, etc., are channels. Either side may open a channel. Multiple channels are multiplexed into a single connection.

Channels are identified by numbers at each end. The number referring to a channel may be different on each side. Requests to open a channel contain the sender’s channel number. Any other channel−related messages contain the recipient’s channel number for the channel.

Channels are flow−controlled. No data may be sent to a channel until a message is received to indicate that window space is available.

5.1. Opening a Channel

When either side wishes to open a new channel, it allocates a local number for the channel. It then sends [a] message to the other side, and includes the local channel number and initial window size in the message. [...]

The remote side then decides whether it can open the channel, and responds with either SSH_MSG_CHANNEL_OPEN_CONFIRMATION or SSH_MSG_CHANNEL_OPEN_FAILURE.

```
byte      SSH_MSG_CHANNEL_OPEN_CONFIRMATION
uint32    recipient channel
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
....      channel type specific data follows
```

The ’recipient channel’ is the channel number given in the original open request, and ’sender channel’ is the channel number allocated by the other side.

```
byte      SSH_MSG_CHANNEL_OPEN_FAILURE
uint32    recipient channel
uint32    reason code
string    description in ISO−10646 UTF−8 encoding [RFC3629]
string    language tag [RFC3066]
```

If the recipient of the SSH_MSG_CHANNEL_OPEN message does not support the specified ’channel type’, it simply responds with SSH_MSG_CHANNEL_OPEN_FAILURE. The client MAY show the ’description’ string to the user. If this is done, the client software should take the precautions discussed in [SSH−ARCH].

The SSH_MSG_CHANNEL_OPEN_FAILURE ’reason code’ values are defined in the following table. [...]

Symbolic name	reason code
SSH_OPEN_ADMINISTRATIVELY_PROHIBITED	1
SSH_OPEN_CONNECT_FAILED	2
SSH_OPEN_UNKNOWN_CHANNEL_TYPE	3
SSH_OPEN_RESOURCE_SHORTAGE	4

[...]

5.2. Data Transfer

The window size specifies how many bytes the other party can send

before it must wait for the window to be adjusted. Both parties use the following message to adjust the window.

```
byte      SSH_MSG_CHANNEL_WINDOW_ADJUST
uint32    recipient channel
uint32    bytes to add
```

After receiving this message, the recipient MAY send the given number of bytes more than it was previously allowed to send; the window size is incremented. Implementations MUST correctly handle window sizes of up to 2^32 − 1 bytes. The window MUST NOT be increased above 2^32 − 1 bytes.

Data transfer is done with messages of the following type.

```
byte      SSH_MSG_CHANNEL_DATA
uint32    recipient channel
string    data
```

The maximum amount of data allowed is determined by the maximum packet size for the channel, and the current window size, whichever is smaller. The window size is decremented by the amount of data sent. Both parties MAY ignore all extra data sent after the allowed window is empty.

Implementations are expected to have some limit on the SSH transport layer packet size (any limit for received packets MUST be 32768 bytes or larger, as described in [SSH−TRANS]). The implementation of the SSH connection layer

- o MUST NOT advertise a maximum packet size that would result in transport packets larger than its transport layer is willing to receive.
- o MUST NOT generate data packets larger than its transport layer is willing to send, even if the remote end would be willing to accept very large packets.

[...]

5.3. Closing a Channel

When a party will no longer send more data to a channel, it SHOULD send SSH_MSG_CHANNEL_EOF.

```
byte      SSH_MSG_CHANNEL_EOF
uint32    recipient channel
```

No explicit response is sent to this message. However, the application may send EOF to whatever is at the other end of the channel. Note that the channel remains open after this message, and more data may still be sent in the other direction. This message does not consume window space and can be sent even if no window space is available.

When either party wishes to terminate the channel, it sends SSH_MSG_CHANNEL_CLOSE. Upon receiving this message, a party MUST send back an SSH_MSG_CHANNEL_CLOSE unless it has already sent this message for the channel. The channel is considered closed for a party when it has both sent and received SSH_MSG_CHANNEL_CLOSE, and the party may then reuse the channel number. A party MAY send SSH_MSG_CHANNEL_CLOSE without having sent or received SSH_MSG_CHANNEL_EOF.

```
byte      SSH_MSG_CHANNEL_CLOSE
uint32    recipient channel
```

This message does not consume window space and can be sent even if no window space is available. [...]