The Movino Protocol

Martin Storsjö

March 2, 2007

Contents

1 Overview

The Movino protocol is a protocol used in the Movino mobile video framework. It is designed as a simple protocol for feeding live video and audio data from a mobile client to a host computer or a remote server. The same protocol can be used both directly from the mobile client and in later links in a stream forwarding chain.

The protocol is assymetrical, in the sense that one peer is denoted server and the other one client. In general, the client initiates the connection to the server, but the protocol would work equally well in a case where the client listens for connections and the server initiates the connection. In a forwarding chain, the forwarding nodes is a server in one connection and a client in the other connections.

2 Data format

The protocol assumes a reliable stream transport protocol (e.g. TCP, RFCOMM). Within the stream, the data is divided into packets. Every packet consists of a 5 byte header and zero or more payload bytes.

The first byte of the header is the packet type byte. The type is interpreted according to in which direction the packet is sent. That is, there are two different lists of packet type numbers; one list for upstream packets (from client to server) and one for downstream packets (from server to client).

The following 4 bytes of the header is the payload size as a unsigned 32-bit integer[1].

The packet size, and all other integers defined in the protocol, are stored in big-endian order, unless something else explicitly is said.

The 5 byte header is immediately followed by the data packet payload, with a length defined in the header. Packets with a payload size of zero therefore only consist of a header.

Since the length of the packet is sent before the payload, there's no need for attaching any special meaning to some bytes or byte combinations, and thus no escaping is needed for the data sent in the payload.

Closing of the connection need not be indicated at the protocol level, closing the underlying transport layer connection is enough. If a packet is only partially received before the transport layer connection is closed, the packet is ignored.

If a packet of an unknown type (e.g. from a newer protocol version) is received, it should be ignored.

2.1 Data format implications

The packet format implies that the current data packet must be sent as a whole before the next packet can be sent. If larger amounts of data is to be sent, one might want to split it into many smaller packets (possibly using some internal data framing information within the payload of the packets).

[1] The current implementations of clients and servers have built-in packet size limits, sizes exceeding the limit is regarded as a bad peer.

Name	Value
Garbage	0
YUV frame	1
PCM audio	2
JPEG header	3
JPEG frame	4
μlaw audio	5
AMR audio	6
Start MTU measurement	7
Stop MTU measurement	8
Video file	9
MPEG frame	10
Video chunk	11
Theora header	12
Theora frame	13
Handshake	14
Stream info	15
Stream closed	16

Table 1: The upstream packet type codes

3 Upstream packets

The upstream packet types and their values are listed in table 1.

3.1 Video packets

The four first payload bytes of packets containing actual video frames contain a 32 bit timestamp for the frame. For the interpretation and usage of these, see section 5.

No specific video frame rate is assumed. Video frames can be sent at any rate feasible, which allows the video stream to adapt to connections of varying bandwidth, e.g. by leaving out later frames if the last one hasn't been transmitted completely. This can easily be applied in the client producing the stream before the frames are encoded.

It can also be applied later in a forwarding node on frame types which don't depend on earlier frames (e.g. JPEG and uncompressed YUV). If formats with inter-frame dependencies (as MPEG and Theora) are used, the received frames must be decoded and the frames decided to be sent must be reencoded again (in any feasible format).

3.1.1 YUV frame

The four first bytes are the timestamp, followed by the frame width as a 16-bit integer, followed by the frame height as a 16-bit integer. The width and height must be even numbers.

After this packet-internal header comes the actual frame data, in YUV420p format. That is, first comes the whole Y-plane of the frame, then the U-plane and last the V-planes. The rows of the planes are stored consecutively with

no padding bytes inbetween, each sample in the planes is 8 bits. The U- and V-planes are downsampled to half the width and half the height of the Y-plane.

The size of the YUV420p data must be at least $width \cdot height + 2 \cdot (width/2) \cdot (height/2)$, any additional data after that is ignored.

3.1.2 JPEG header

This packet consists of JPEG data for setting JPEG decoding parameters. It contains no frame data.

3.1.3 JPEG frame

The first 4 bytes contains the timestamp, followed by the an ordinary JPEG file. It is decoded in the same context as earlier JPEG frames and JPEG header packets. Therefore, the JPEG frames can be abbreviated JPEG streams, containing no tables, but in this case, the tables must have been decoded in earlier JPEG frame or header packets.

3.1.4 Video file

This packet only contains a complete video file (containing both video and audio) in an unspecified format, which must be identified by the decoder.[2]

3.1.5 MPEG frame

The first 4 bytes contains the timestamp. The rest of the packet is a raw MPEG1-encoded frame. The frame is decoded in the same context as earlier MPEG frames.

3.1.6 Video chunk

Not used anymore, deprecated.

3.1.7 Theora header

This packet contains the headers for initializing a Theora decoder. The packet payload is scanned for one or more Ogg pages, which may contain one or more Ogg stream packets. All these Ogg stream packets are fed to the header decoding routines of a Theora decoder.

3.1.8 Theora frame

The first 4 bytes contains the timestamp. The rest of the packet is scanned for one or more Ogg pages, which may contain one or more Ogg stream packets. All Ogg stream packets are fed to the content decoding routines of a Theora decoder, which try to decode YUV frames from the content.

[2]Support for this packet is implemented only in very small parts of the framework. Don't use it.

3.2 Audio packets

The four first payload bytes of all audio data packets contain a 32 bit timestamp for the frame. For the interpretation and usage of these, see section 5.

3.2.1 PCM audio

The first 4 bytes contains the timestamp. The rest of the packet contains one or more 16-bit audio samples. The audio samples are stored as consecutive signed 16-bit integers in little-endian format. The sampels belong to one mono-recording at 8000 Hz. The payload size must be even.

3.2.2 μlaw audio

The first 4 bytes contains the timestamp. The audio data content is equivalent to the content in PCM audio packets, except that the individual samples are stored as 8-bit μlaw encoded samples. The audio data content size thus is half the size of a corresponding PCM audio packet.

3.2.3 AMR audio

The first 4 bytes contains the timestamp. The rest of the payload is one single AMR-NB packet. The packets are decoded in the same context as earlier packets.

3.3 Other packets

3.3.1 Start MTU measurement

This packet contains no payload. It tells the server to prepare for a MTU measurement.

3.3.2 Stop MTU measurement

This packet contains no payload. It tells the server that the MTU measurement is finished, and the server should reply with the size of the largest transport layer packet received during the measurement. See the packet format for the MTU size downstream packet in 4.1.

3.3.3 Handshake

See the the section on the handshake procedure, section 6.

3.3.4 Stream info

This packet contains metadata on the contents of the stream.

First comes the width of the video in the stream, as a 16-bit integer, then the height as a 16-bit integer. These are only hints about the size, the actual video frames can be of any size and can change between frames without any stream info packet.

Then comes an 8-bit integer specifying whether the stream contains audio. The integer is interpreted as a boolean value, where zero is false and any nonzero

Name	Value
MTU size	1
Handshake	2
Ping	3

Table 2: The downstream packet type codes

value is true. This can be used as a hint to whether the server should try to sync the image data to audio data.

After audio info byte comes a human-readable string with the name of the content author. It serves no purpose in the actual protocol, but can be used by the server in any way suitable. Then comes a title of the content stream, in the same fashion. Both strings are preceded by a 16-bit integer denoting the length of the string (excluding these two size bytes), followed by the string encoded in UTF-8. The strings don't contain null termination.

After the strings comes one single byte interpreted as a boolean, specifying whether the stream content should be archived.

Any further data in the packet is ignored (which could be used to add more metadata in later protocol versions).

3.3.5 Stream closed

This packet contains no payload. It tells the server that the current content stream (consisting of consecutive audio and/or video packets) has ended, even though the transport layer connection still is open. This is appropriate e.g. when the client of a forwarding node disconnects, but the forwarder still keeps the forwarding connection open for future use.

3.3.6 Garbage

The garbage packet can be used for testing communication link capabilities. The content is ignored.

4 Downstream packets

The downstream packet types and their values are listed in table 2.

4.1 MTU size

The payload consists of a 32-bit integer denoting the size of the largest transport layer packet received during the measurement period. It is sent as a reply to a Stop MTU measurement packet.

4.2 Handshake

See the the section on the handshake procedure, section 6.

Username types:

Name	Value
No username required	1
Username required	2

Password types:

Name	Value
No password required	1
MD5 challenge/response	2

Table 3: The handshake constants

4.3 Ping

This packet contains no payload. It is sent by the server when it haven't received any data at all for some time. The client should send some data (e.g. an empty garbage packet) as soon as possible. The server may close the transport layer connection if the peer doesn't send any data in time.[3]

This packet is sent only once when inactivity is noticed. No resending is needed, since that is handled by the underlying transport layer protocol. If the transport layer protocol can't transport these messages on time, the connection can be closed (even though the client would be alive), since the connection is useless anyway in that case.

5 Timestamps

The timestamps in the audio and video packets are used to allow for syncrhonization between the audio and video data. They may also be used for e.g. displaying the received video frames at an even rate even if the connection can't transfer them at an even rate.

The timestamps are 32-bit integers storing the time of the video frame or audio block since an arbitrary earlier point in time, in milliseconds. The timestamps need not start from 0 in the beginning of the stream.

This allows the audio and video packets to be forwarded unmodified, even though the following server haven't received all packets from the beginning of the stream.

6 Handshake procedure

When the transport layer connection is established, the server sends a downstream handshake packet to the client. The client must reply with a valid upstream handshake packet accordingly, otherwise the server closes the connection. If any other packet is received by the server before the handshake is done, the connection is closed.

[3] In the current implementation, the ping packet is sent 10 seconds after the last received data, and if nothing is received in 30 seconds (since the last received data, not since the the ping packet is sent), the connection is closed.

The initial downstream handshake packet consists of two bytes saying if any password and username is required to connect. The first byte describes the username and the second the password. Constants for these bytes are defined in table 3. A username can only be requested if a password is requested.

If the password type byte is set to MD5 challenge, the rest of the payload is a block of random data used as a challenge.

The upstream handshake reply starts with the same two bytes as received from the server. If no password is requested, the payload consists of nothing more. If a password is required, the rest of the packet starts with the username string (which is ignored but must be present if no username was requested). The string starts with a 16-bit integer telling the length of the rest of the string, followed by the actual string (encoded as UTF-8). After the username string comes the password response, preceded by a 16-bit integer denoting the length of the response.

If the password is requested as a MD5 challenge response, the response is 16 bytes calculated in this way:

$$\text{MD5}(\text{MD5}(\text{password}) + \text{challenge})$$

That is, the UTF-8-encoded password is hashed with the MD5 hash function. The resulting (binary) hash is concatenated with the challenge received from the server, and hashed once again. The initial MD5 hashing of the password alone is added to enable integration with the Drupal user database, which only stores the MD5 hashes of users' passwords.

After the client has sent the handshake reply, it assumes that the response was accepted, and can start sending any other packet immediately. If the response wasn't accepted, the server simply closes the transport layer connection.

7 Transport layer conventions

When the Movino protocol is used over RFCOMM (over Bluetooth), the server should advertise the RFCOMM channel it is listening to using the SDP[4] 32-bit UUID 0x1027392E.

When used over TCP, the server could be listening on TCP port 30710 by default.

[4]Service Discovery Protocol