Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 September 2022

C. Jennings
cisco
R. Logan
Cisco

4 March 2022

Game State over Real Time Protocol
draft-jennings-dispatch-game-state-over-rtp-00

Abstract

This specification defines an Real Time Protocol (RTP) payload to send game moves and the state of game objects over RTP. This is useful for games as well collaboration systems that use augment or virtual reality.

RTP provide a way to synchronize game state between players with robust technique for recovery from network packet loss while still having low latency.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 5 September 2022.

Copyright Notice

Table of Contents

1. Overview

Many real time applications, such as games, want to to share state about 3D objects across the network. This specification allows an application to define objects with state, and the current values of that state over RTP.

The conceptual model is each RTP sender has a small number of objects with state that needs to be synchronized to the other side. The current values are periodically sent over RTP. They MAY be sent when the values change, but they MUST also be sent periodically so that any lost updates are eventually received and state is consistent between the sender and receiver.

The state sent can include a time stamp and rate change estimates that allow the receiver to estimate the current state values even at a point in the future. An application that receives a state update can apply it immediately (often called immediate based), wait a fixed delay and then apply state changes (often called delay based), or apply a predicted value based on overwriting any previous predictions (often called rollback based).

In many cases the state does not have any units but if does, SI units SHOULD be used. Unless otherwise defined by the application, the default coordinate system SHOULD be a left handed coordinate system where Y points up and X points to the right.

Applications can define their own objects or use some of the predefined common objects. Each object is identified by a type and identifier number to uniquely identify the object within the scope of that sender's RTP stream. Multiple updates and objects can be combined in a single RTP packet so that they are guaranteed to be fate shared and either atomically delivered at the same time or not delivered at all to the receiver.

The objects are defined as a series of primitives that define common types. The objects and updates to state are encoded with Tag Length Value (TLV) style encoding so that receivers can skip objects they do not understand. The Objects in an single RTP packet MUST be processed in order. This allows a sender to write state in an old format followed by a new format, allowing the new format to override values in the old format. This allows for easy upgrade of the protocol with backwards compatibility.

2. Goals:

* Support 2D and 3D
* Support delay and rollback based synchronization

* Relatively compact, simple encoding
* Extensible for applications to send custom data
* Support for forward and backwards compatibility

3. Primitives

This section defines primitives that are useful in defining objects. The definition are in W3C style EBNF [https://www.w3.org/TR/2010/REC-xquery-20101214/#EBNFNotation (https://www.w3.org/TR/2010/REC-xquery-20101214/#EBNFNotation)].

3.1. Location

```
Loc1 ::=
 Float32 /* x */
 Float32 /* y */
 Float32 /* z */
```

Loc1 is simply a 3D location of a point stored in Float32;

```
Loc2 ::=
 Float32 /* x */
 Float32 /* y */
 Float32 /* z */
 Float16 /* vx */
 Float16 /* vy */
 Float16 /* vz */
```

Loc2 has a location as Float32 followed by the rate of change in location per second as Float16.

3.2. Scale

```
Scale1 ::=
Float16 /* all dimensions */
```

Scale1 will scale the object in all dimensions equally.

```
Scale2 ::=
 Float32 /* x */
 Float32 /* y */
 Float32 /* z */
 Float16 /* vx */
 Float16 /* vy */
 Float16 /* vz */
```

Scale2 has a scale in each axis as Float32 followed by the rate of change in the scale per second as Float16.

3.3. Normal

```
Norm1 ::=
 Float16 /* nx */
 Float16 /* ny */
 Float16 /* nz */
```

Normal vector for a point.

3.4. TextureUV

```
TextureUV1 ::=
 VarUInt /* u */
 VarUInt /* v */
```

Location in texture map for a point.

3.5. Rotation

```
Rot1 ::=
 Float16 /* i */
 Float16 /* j */
 Float16 /* k */
```

The non-real parts of a normalized rotation quaternion. The real part can be computed based on how it is normalized.

```
Rot2 ::=
 Float16 /* s.i */
 Float16 /* s.j */
 Float16 /* s.k */
 Float16 /* e.i */
 Float16 /* e.j */
 Float16 /* e.k */
```

Rot2 defines the s, the current rotation, and e, an estimated value of the rotation in one second. The rotation e is chosen such that rotation of the object will follow a rotation along the great circle path from s to e with an constant angular rotation rate such that it would reach e in 1 second.

This representation of rate of change of the rotation allows the receiver to use an algorithm such as SLERP [https://en.wikipedia.org/wiki/Slerp (https://en.wikipedia.org/wiki/Slerp)] to estimate the current rotation for the object for short periods of time into the future. The representation cannot represent something that is rotating faster than one revolution ever 2 seconds.

Open Issue: Would there be a better way to represent angular velocity?

3.6. Child Transform

```
Transform1 ::=
 Float16 /* tx */
 Float16 /* ty */
 Float16 /* tz */
```

Defines a linear translation of a child object from a base object.

3.7. Texture URL

```
TextureUrl1 ::= String
```

URL of image with texture map. JPEG images SHOULD be supported.

3.8. Mesh URL

```
MeshUrl1 ::= String
```

URL of external mesh.

Open Issue: Any mandatory to implements mesh formats?

3.9. Texture Stream

In some cases it is desirable to provide a separate RTP video stream which might have the texture used be the frame from the video stream with the corresponding time to the game object that uses the texture map. To do this there needs to be a way to identity the other RTP video stream. One way to do that is use the "Payload Type" value used for the RTP packets for that video stream.

```
TextureRtpPT1 ::= UInt8 /* pt */
```

RTP "Payload Type" value of RTP video stream to use as a texture map.

Open Issue: Is there a better way to identity video stream?

3.10. Timestamp

```
Time1 ::= UInt16 /* time in ms */
```

Lower 16 bits of number in milliseconds since 00:00:00 UTC on 1 January 1970, not counting leap seconds. The assumption is these timestamps are accurate to about the level of an NTP synchronized clock. In C++20 this can be found with:

```
duration_cast<milliseconds>(
      system_clock::now().time_since_epoch()
    ).count() % 65536
```

4. Objects

All objects must start with a unique tag that defines the object type, a VarUInt length, a VarUInt objectID, and the data for object. Applications can reserve tags for objects in the registry defined in the IANA section. Objects can be made extensible by adding a section that contains optional tag, length, value tuples.

4.1. Common Objects

4.1.1. Generic Game Object

It is common to need to describe the location, rotation, scale, and parent for objects in a scene.

```
Object1 ::= tagObject1 Length ObjectID Time1
 Loc1
 Rot1
 Scale1
  ( tagParent1 Length ObjectID )? /* Optional Parent */
```

Object1 contains a simple 3D location, rotation, scale and an optional ID of a parent object.

```
Object2 ::= tagObject2 Length ObjectID Time1
 Loc2
 Rot2
 Scale2
 ( tagParent1 Length ObjectID )? /* Optional Parent */
```

Object2 has the same information but with the ability to scale differently in each dimension and derivatives that describe how all the parameters change over time.

4.1.2. Player Head

```
Head1 ::= tagHead1 Length ObjectID Time1
  Loc2 Rot2
  ( tagHeadIpd Length Float16 /* IPD */ )?
```

Defines location and rotation of head with optional interpupillary distance (IPD).

4.1.3. Mesh

Object with the following variable length arrays:

```
Mesh1 ::= tagMesh1 Length ObjectID
 ( TextureUrl1 | TextureRtpPT1 )
 VarUInt /* num Vertexes */
 Loc1+ /* vertexes */
 VarUInt /* numNormals */
 Norm1* /* normals */
 VarUInt /*  numTextureCoord */
 TextureUV1* /*  textureCoord */
 VarUInt /* numTrianglesIndex */
 VarUInt+ /* trianglesIndex */
```

The vertex is an array of at least 3 locations that defines the vertex of a triangle mesh. The normals array can either be empty or the same size as the vertex and defines the normal for each vertex. The uv array must be empty or the same size as vertex array and have the u,v coordinate in the texture map for the vertex.

The texture can be defined by a URL that may refer to some local resource or a resource retrieved over the network. Alternatively, the texture can reference a local RTP video stream, in which case the most recently received frame of video is used as the texture and texture updates with new frames of video.

The triangles array can be of a different size from the vertex array. Each entry defines one triangle in the mesh and contains the index of the three vertexes in the vertexes array. Vertexes MUST be in counter clockwise order.

An important limitation to note is that objects cannot span RTP packets so the Mesh needs to be small enough that it size is less that the MTU. A typical limit might be as low as 50 triangles.

4.1.4. External Mesh

The Mesh2 object allows a mesh to be loaded from an external URL and then moved, rotated, and scaled. An optional texture map may be used.

```
Mesh2::= tagMesh2 Length ObjectID
  Loc2 Rot2 Scale2
  MeshUrl1
  ( TextureUrl1 | TextureRtpPT1 )? /* Optional Texture Map */
  ( tagParent1 Length ObjectID )? /* Optional Parent */
```

4.1.5. Player Hand

```
Hand1 ::= tagHand1 Length ObjectID Time1
 Boolean /* left */
 Loc2 Rot2
```

The Hand1 identifies a location and rotation of a hand. The left is true for the left hand, false for a right hand.

```
Hand2 ::= tagHand2 Length ObjectID Time1
 Boolean /* left */
 Loc2 Rot2
 Transform1 /* wrist */
 Transform1 /* thumbTip */
 Transform1 /* thumbIP */
 Transform1 /* thumbMCP */
 Transform1 /* thumbCMC */
 Transform1 /* indexTip */
 Transform1 /* indexDIP */
 Transform1 /* indexPIP */
 Transform1 /* indexMCP */
 Transform1 /* indexCMC */
 Transform1 /* middleTip */
 Transform1 /* middleDIP */
 Transform1 /* middlePIP */
 Transform1 /* middleMCP */
 Transform1 /* middleCMC */
 Transform1 /* ringTip */
 Transform1 /* ringDIP */
 Transform1 /* ringPIP */
 Transform1 /* ringMCP */
 Transform1 /* ringCMC */
 Transform1 /* pinkyTip */
 Transform1 /* pinkyDIP */
 Transform1 /* pinkyPIP */
 Transform1 /* pinkyMCP */
 Transform1 /* pinkyCMC */
```

Hand2 represents a wired skeletal hand. The boolean is true for the left hand. The location should represent the location of the palm and rotation is from the palm facing the x axis.

The transform points represent the relative location to the main joints in the fingers from the hand location. The location of the wrist is followed by finger joints. The fingers are ordered by thumb, index, middle, ring, then pinky. The joints are ordered by tip of finger (TIP), distal interphalangeal joint (DIP), proximal interphalangeal joint (PIP), metacarpophalangeal joint (MCP), then carpometacarpal joint (CMC). Note the thumb has no middle phalange so the PIP and DIP joint just become the interphalangeal joint (IP).

Note: The Microsoft documentation calls the MCP "knuckle" for the fingers and PIP "middle joint" and the CMC is called "Metacarpal", which is very confusing since this is not the metacarpophalangeal joint. The MCP for the thumb gets called "proximal", not knuckle, and the IP is "middle".

Names of the joints are explained in [https://en.wikipedia.org/wiki/Interphalangeal_joints_of_the_hand (https://en.wikipedia.org/wiki/Interphalangeal_joints_of_the_hand)]

This is about 175 bytes, so at a 5Hz update rate this will be around 10 Kbps. [TODO: Check.]

5. Encoding

Each RTP payload will contain one or more objects. An object cannot be split across two RTP packets. The general design is that if the decoder has not been coded to understand a given object type, the decode can skip over the object to the next object but will not be able to provide any information and the internal format of the data.

The objects are defined such that they always start with a tag that indicates the type followed by a length of the object (so it can be skipped). Any optional or variable parts of the object also use tags so that the decoder can always be implemented as a LL(1) parser.

In general, network byte order encoding is used on the wire.

A length field is encoded to represent the number of bytes following the length field and does not include the size of the length field or information before it.

5.1. Tag

Constant values of tags can be found in the IANA section. They are encoded as VarUInt.

5.2. Float

Float16, Float32, and Float64 are encoded as IEEE 754 half, single, and double precisions respectively.

The half precision are often useful for things where only a few significant digits are needed such as normals. The internal representation of them will often be single precession (four bytes) in memory but they can be reduced to two bytes when encoded on the wire.

Note there is an example decode for a single precision float at [https://datatracker.ietf.org/doc/html/rfc7049#appendix-D (https://datatracker.ietf.org/doc/html/rfc7049#appendix-D)]

5.3. Boolean

Encoded as byte with 0 for false or 1 for true.

5.4. Integer

UInt8, Int8, UInt16, Int16, UInt32, Int32, UInt16, Int64 encoded as 1, 2, 4, or 8 bytes.

VarInt is encoded as:

* Top bits of first byte is 0, then 7 bit signed integer (-64 to 63)

* Top bits of first byte is 10, then 6+8 bit signed integer (-8192 to 8191)

* Top bits of first byte is 110, then 5+16 bit signed integer (1,048,576 to 1,048,575)

* Top bits of first byte is 1110,0001 then next 4 bytes 32 bit signed integer

* Top bits of first byte is 1110,0010 then next 8 bytes 64 bit signed integer

VarUInt is encoded as:

* Top bits of first byte is 0, then 7 bit unsigned integer

* Top bits of first byte is 10, then 6+8 bit unsigned integer

* Top bits of first byte is 110, then 5+16 bit unsigned integer

* Top bits of first byte is 1110,0001 then next 4 bytes 32 bit unsigned integer

* Top bits of first byte is 1110,0010 then next 8 bytes 64 bit unsigned integer

5.5. String

Strings are encoded as a VarUInt length in bytes (not characters) followed by a UTF-8 representation of the string.

5.6. Blob

Blobs are encoded as a VarUInt length in bytes followed by the binary data that goes in the blob.

6. Full Intra Request

RTP supports a Full Intra Request (FIR) Feedback Control feedback messages. When an RTP sender receives a FIR, it SHOULD send a copy of all the relevant game state.

7. IANA

8. RTP

This section can be split out a separate payload draft we need some extra work.

The media type is application/gamestate. There are no optional or required parameters. The RTP marker bit is not used. The RTP clock MUST be 90 kHz.

Multiple Objects as defined in this specification can be concatenated into one RTP payload.

TODO: The SDP MAY include an objectTags type that indicates the tag values of all the supported objects types.

TODO: define storage format as well as RTP payload format details.

8.1. Game State Tag Registry

The specification defines a new IANA registry for tag values. All values MUST be greater than zero.

Values 1-127 are assigned by "IETF Review" as defined in [RFC8126], and should only be used when size is critical, the object is small, and will be used frequently.

Values 127-16383 are assigned by "Specification Required" as defined in [RFC8126].

Values 16384 to 2,097,151 are "First Come First Served" as defined in [RFC8126].

Initial assignments are:

TagName	Value
tagInvalid	0
tagHead1	1
tagHand1	2
tagObject1	3
tagParent1	4
tagMesh1	128
tagHand2	129
tagHeadIPD1	130
tagObject2	131
tagMesh2	132

Table 1

9. Security

Like most things in RTP, the data can be personal identifying information. For example, the Hand2 type of data when generated by tracking a persons hand might identify that user.

Appendix A. Acknowledgments

Thanks to Paul Jones for comments and writing an implementation.

Appendix B. Implementations

A C++ open source implementation is available at: TODO.

Appendix C. Test Vectors

C.1. Head Location

Head Location type 1 with head at location 1.1, 0.2, 30.0, no rotation (so quaternion 0, 0, 0, 1) and not rotating, an objectID of 4, a time of 5 ms after epoch and an IPD of 0.056.

Field	Type	Value	Hex
head1	Tag	1	0x01
len	VarInt	33	0x21
objID	VarInt	0	0x00
time	UInt16	5	0x0500
loc.x	Float32	1.1	0x3F8CCCCD
loc.y	Float32	0.2	0x3E4CCCCD
loc.z	Float32	30.0	0x41F00000
loc.vx	Float16	0.0	0x0000
loc.vy	Float16	0.0	0x0000
loc.vz	Float16	0.0	0x0000
rot.s.i	Float16	0.0	0x0000
rot.s.j	Float16	0.0	0x0000
rot.s.k	Float16	0.0	0x0000
rot.e.i	Float16	0.0	0x0000
rot.e.j	Float16	0.0	0x0000
rot.e.k	Float16	0.0	0x0000

Table 2

Appendix D. Encode API

API that take a high level representation of each object where types are all float or int and returns a memory buffer.

Appendix E. Decode API

API that takes binary data and set up objects and updates the objects and returns object ids that were updated.

For each object, an API to get the predicted values at a given time.

Appendix F. EBNF

```
Head1 ::= tagHead1 Length ObjectID Time1 Loc2 Rot2
  ( tagHeadIpd Length Float16 /* IPD */ )?

Mesh1 ::= tagMesh1 Length ObjectID
 ( TextureUrl1 | TextureRtpPT1 )
 VarUInt /* num Vertexes */
 Loc1+ /* vertexes */
 VarUInt /* numNormals */
 Norm1* /* normals */
 VarUInt /*  numTextureCoord */
 TextureUV1* /*  textureCoord */
 VarUInt /* numTrianglesIndex */
 VarUInt+ /* trianglesIndex */

Mesh2::= tagMesh2 Length ObjectID
  Loc2 Rot2 Scale2
  MeshUrl1
  ( TextureUrl1 | TextureRtpPT1 )? /* Optional Texture Map */
  ( tagParent1 Length ObjectID )? /* Optional Parent */

Object1 ::= tagObject1 Length ObjectID Time1
 Loc1
 Rot1
 Scale1
  ( tagParent1 Length ObjectID )? /* Optional Parent */

Object2 ::= tagObject2 Length ObjectID Time1
 Loc2
 Rot2
 Scale2
 ( tagParent1 Length ObjectID )? /* Optional Parent */

Hand1 ::= tagHand1 Length ObjectID Time1
 Boolean /* left */
 Loc2 Rot2

Hand2 ::= tagHand2 Length ObjectID Time1
 Boolean /* left */
 Loc2 Rot2
 Transform1 /* wrist */
 Transform1 /* thumbTip */
 Transform1 /* thumbIP */
 Transform1 /* thumbMCP */
 Transform1 /* thumbCMC */
```

```
    Transform1 /* indexTip */
    Transform1 /* indexDIP */
    Transform1 /* indexPIP */
    Transform1 /* indexMCP */
    Transform1 /* indexCMC */
    Transform1 /* middleTip */
    Transform1 /* middleDIP */
    Transform1 /* middlePIP */
    Transform1 /* middleMCP */
    Transform1 /* middleCMC */
    Transform1 /* ringTip */
    Transform1 /* ringDIP */
    Transform1 /* ringPIP */
    Transform1 /* ringMCP */
    Transform1 /* ringCMC */
    Transform1 /* pinkyTip */
    Transform1 /* pinkyDIP */
    Transform1 /* pinkyPIP */
    Transform1 /* pinkyMCP */
    Transform1 /* pinkyCMC */

  Tag ::= VarUInt

  tagInvalid ::= #x00
  tagHead1 ::= #x01
  tagHand1 ::= #x02
  tagObject1 ::= #x03
  tagParent1 ::= #x04
  tagMesh1 ::= #x80 #x00
  tagHand2 ::= #x80 #x01
  tagHeadIpd ::= #x80 #x02
  tagMesh2 ::= #x80 #x04
  tagObject2 ::= #x80 #x03

  ObjectID ::= VarUInt

  Length ::= VarUInt

  Loc1 ::=
    Float32 /* x */
    Float32 /* y */
    Float32 /* z */

  Loc2 ::=
    Float32 /* x */
    Float32 /* y */
    Float32 /* z */
    Float16 /* vx */
```

```
    Float16 /* vy */
    Float16 /* vz */

   Scale1 ::=
    Float16 /* all dimensions */

   Scale2 ::=
    Float32 /* x */
    Float32 /* y */
    Float32 /* z */
    Float16 /* vx */
    Float16 /* vy */
    Float16 /* vz */

   Norm1 ::=
    Float16 /* x */
    Float16 /* y */
    Float16 /* z */

   TextureUV1 ::=
    VarUInt /* u */
    VarUInt /* v */

   Rot1 ::=
    Float16 /* i */
    Float16 /* j */
    Float16 /* k */
    /* w computed based on quaternion is normalized */

   Rot2 ::=
    Float16 /* s.i */
    Float16 /* s.j */
    Float16 /* s.k */
    Float16 /* e.i */
    Float16 /* e.j */
    Float16 /* e.k */

   Transform1 ::=
    Float16 /* tx */
    Float16 /* ty */
    Float16 /* tz */

   TextureUrl1 ::= String

   MeshUrl1 ::= String

   TextureRtpPT1 ::= UInt8 /* pt */
```

```
   Time1 ::= UInt16 /* time in ms */

   Tag ::= VarUInt

   Boolean ::=  #x00 | #x01

   String ::= VarUInt byte*

   Blob ::= VarUInt byte*

   Float16 ::= byte byte
   Float32 ::= byte byte byte byte
   Float64 ::= byte byte byte byte byte byte byte byte

   Int8 ::= byte
   Int16 ::= byte byte
   Int32 ::= byte byte byte byte
   Int64 ::= byte byte byte byte byte byte byte byte

   UInt8 ::= byte
   UInt16 ::= byte byte
   UInt32 ::= byte byte byte byte
   UInt64 ::= byte byte byte byte byte byte byte byte

   VarUInt ::=
    ( [#x0-#x7F] ) |
    ( [#x80-#x87] byte ) |
    ( [#x88-#x8B] byte byte ) |
    ( #xE1 UInt32 ) |
    ( #xE2 UInt64 )

   VarInt ::=
    ( [#x0-#x7F] ) |
    ( [#x80-#x87] byte ) |
    ( [#x88-#x8B] byte byte ) |
    ( #xE1 Int32 ) |
    ( #xE2 Int64 )

   byte ::= [#x00-#xFF]
```

Authors' Addresses

Cullen Jennings
cisco
Canada
Email: fluffy@iii.ca

Rich Logan
Cisco
United Kingdom
Email: rilogan@cisco.com