INTERNET DRAFT 10 March 2000
Expires: 10 September 2000

Sean Sheedy
nCUBE Corporation

RTSP Extensions:
Additional Transports and Performance Enhancements

draft-sheedy-mmusic-rtsp-ext-00.txt

Status of this memo

Abstract

This document proposes enhancements to the RTSP protocol for broadcast quality non-IP based video-on-demand applications. Additional transports for non-IP delivery of media streams are proposed, along with control extensions to reduce latency.

These proposals are based on nCUBE Corporation's and Oracle Corporation's experience with their existing media servers.

1 Introduction

nCUBE Corporation has developed a media server using the RTSP standard [1] for its video-on-demand (VOD) platform, the MediaCUBE 4. The platform is designed for large-scale deployments of broadcast quality interactive video. It is being used currently in several commercial deployments worldwide, with many more deployments scheduled in the near future.

nCUBE's experience to date with the RTSP protocol has been positive. The basic protocol is flexible enough to work in a large-scale, high-bandwidth environment. The HTTP-like syntax has proven easy for client developers to implement. The flexibility provided by the syntax and the facilities for extensions have proven invaluable in deploying RTSP in an environment somewhat different from that for which it was originally designed.

A typical broadcast quality environment differs from the Internet environment in several ways:

- Transports and lower transports

  Although IP protocols are often used for control connections, broadcast quality video-on-demand installations often do not use IP protocols (such as UDP and RTP) for the actual delivery of a presentation (which is usually an aggregation of media streams). Typically, MPEG-2 transport streams are carried on a lower transport which natively supports MPEG-2, such as AAL5 (over ATM) or QAM.

- Multiplexing RTSP clients

  Supporting non-RTSP clients (e.g., many currently-available set top boxes) requires a bridging server that speaks the client's native protocol and, in turn, acts as an RTSP client of the media server. Such bridging servers typically make transport address and bandwidth assignments for the clients, and often need to coordinate these decisions with external hardware devices such as QAM modulators and up converters.

- Limited capability clients

  Video-on-demand clients in the home (such as specialized set top boxes) are under tremendous price pressures. Consequently, their capabilities are often much more limited than even low-end general-purpose computers. Memory is typically very limited (on the order of 8 megabytes), and the media streams are discarded immediately once they have been decoded. Many hardware decoders are sensitive to timing jitter and discontinuities in video or audio elementary streams.

- Latency

  Low latency, particularly for stream control requests such as pause or fast forward, is critical to the satisfaction of many home users of video-on-demand services. End users of a home VOD service are a cross section of cable television customers, and are often not computer savvy. Their standard of comparison for the responsiveness of a media server is their VCR or DVD player, not a computer web browser accessing the Internet.

nCUBE Corporation has added some extensions to the RTSP standard, which address the requirements of this different environment [3]. These enhancements were developed in collaboration with Oracle Corporation, who has also incorporated similar features in their RTSP server [4]. The remainder of this document proposes a set of enhancements to the RTSP standard based on both of these implementations.

2 Transports, Profiles, and Lower Transports

Other media delivery mechanisms besides RTP are used in many commercial video-on-demand deployments. To support this, new transports, profiles, and lower transports in addition to the current standard RTP/AVP/UDP are needed.

2.1 Transports

MPEG-2 is used extensively for high bandwidth video [5]. An enhancement to the "transport-protocol" field in the "Transport" header to support this is:

```
MP2T   MPEG-2 Transport
```

The new syntax of "transport-protocol" would then be:

```
transport-protocol = "RTP" | "MP2T"
```

2.2 Lower Transports

Similarly, TCP or UDP are not always used as lower transports. Enhancements to the "lower-transport" field are:

```
AAL5-PVC   ATM permanent virtual circuit

AAL5-SVC   ATM switched virtual circuit

ASI        DVB Asynchronous Serial Interface (ASI)

QAM        Quadrature Amplitude Modulation(QAM)
```

The new syntax of "lower-transport" would then be:

```
lower-transport = "TCP" | "UDP" | "AAL5-PVC" | "AAL5-SVC" |
                  "ASI" | "QAM"
```


(Note that end user RTSP clients typically don't request a DVB-ASI lower transport. This is primarily used by bridging servers that are also controlling external hardware such as QAM modulators.)

This proposal makes no distinction between QAM64 and QAM256. Making such a distinction may be desirable.

2.3 Profiles

Since AVP is intertwined with RTP, additional profiles are needed. Enhancements to the "profile" file are:

```
H2221   The ITU H.222.1 standard for MPEG delivery over
        ATM [6].  The default lower transport is AAL5-SVC.
```

DVBC   The Digital Video Broadcasting - Cable standard
        [7]. The default lower transport is QAM.

The new syntax of "profile" would then be:

```
profile = "AVP" | "H2221" | "DVBC"
```

2.4 Transport/Profile/Lower-Transport Combinations

Only the following combinations of protocols, profiles, and lower transports are meaningful:

```
MP2T/H2221/AAL5-PVC
MP2T/H2221/AAL5-SVC
MP2T/DVBC/QAM
MP2T/DVBC/ASI
```

2.5 Destinations

To handle the additional lower transport types, the syntax of the "destination" transport parameter needs to be enhanced. In particular, most lower transports in section 2.2 use addresses that are not globally unique, but are unique only within a particular physical channel. The destination "address" field should contain an optional identifier string at the beginning to allow sufficiently intelligent clients (such as bridging servers) to disambiguate between physical channels.

The formal syntax for the expanded "destination" addresses is:

```
address      = [ id-string ":" ] type-address

type-address = host
             | atm-pvc-address
             | atm-svc-address
             | qam-address

id-string    = 1*( ALPHA | DIGIT | "_" )
```


(Note that QAM and DVB-ASI addressing are identical, and both are covered by the "qam-address" rule.)

2.5.1 ATM PVC Address

The destination address for an ATM permanent virtual circuit is the VPI and VCI of the client, specified in decimal:

```
atm-pvc-address = vpi "." vci

vpi                = 1*5(DIGIT)
vci                = 1*5(DIGIT)
```

For example, to use ATM permanent virtual circuits a client may specify a Transport header like the following:

```
Transport: MP2T/H2221/AAL5-PVC;unicast;destination=atm00:0.40
```

2.5.2 ATM SVC Address

The destination address for an ATM switched virtual circuit is the 20-byte service access point address, specified in hexadecimal:

```
atm-svc-address = 20*20(HEX)
```

For example, to use ATM switched virtual circuits a client may specify a Transport header like the following:

```
Transport: MP2T/H2221/AAL5-SVC;unicast;
           destination=47000580ffe1000000f21a360b00204821490f01
```

Many vendors include dots (.) in service access point addresses. It may be desirable to allow this.

2.5.3 QAM and DVB-ASI Addresses

The destination address for QAM or DVB-ASI is a server-specific channel number (note that this is not the RF channel number) and MPEG-2 program number specified in decimal:

```
qam-address    = channel-number "." program-number

channel-number = 1*3(DIGIT)
program-number = 1*5(DIGIT)
```

For example, to use QAM a client may specify a Transport header like the following:

```
Transport: MP2T/DVBC/QAM;unicast;destination=cim00:0.75
```

For example, to use DVB-ASI a client may specify a Transport header like the following:

```
Transport: MP2T/DVBC/ASI;unicast;destination=dac00:0.75
```

2.6 Client Identification

For most video-on-demand environments, clients cannot be allowed to specify a transport destination address. In non-IP delivery environments, they typically do not have sufficient knowledge of the network topology to properly specify an address. In all environments, allowing clients to choose an address presents security problems. Further, in many non-IP delivery environments (such as cable systems using QAM and DOCSIS), valid transport addresses cannot be derived from the IP address of the client.

To resolve these problems, the client must be able to identify itself to the media server. This can be accomplished by adding a new Transport parameter, "client". The argument to "client" is a deployment-specific string that uniquely identifies a client. Identity information included in the string may be, for example, a smart card ID for a set top box and the optical node to which the set top box is connected.

The formal syntax for the "client" parameter is:

```
client    = "client" "=" client-id
client-id = token
```

3 Reuse of Transports

Typically, the setup and teardown of a transport are the most expensive media server operations, both in terms of server loading and client perceived latency. The Web browsing model of creating a transport for each presentation works well in many Internet delivery environments. In a non-IP delivery environment with dedicated media delivery bandwidth, however, using a single transport for several sequential presentations provides a better end user experience.

Allowing a single transport to handle multiple sequential presentations requires extensions in the following areas:

- URI's

- Transport parameters

3.1 URI Enhancements

To allow a single transport to be used for different presentations, the client may specify a different URI on a PLAY method request than was used in the initial SETUP request. If a PLAY is requested with a different URI than that most recently used in the session, the presentation specified by the new URI will be played over the existing session's transport.

For a PLAY request with a new URI to succeed, sufficient bandwidth must already be available in the existing transport. This can be reserved with an extension transport parameter on the initial SETUP of the session ("bandwidth", described in section 3.2), or can be allocated with a new SETUP request.

If a client uses queued PLAY requests with different URI's, it may not be able to determine which presentation is active at any particular time. To handle this case, an asterisk (*) for the URI matches whatever presentation, if any, is currently active. Such a wild card asterisk is legal for the following methods:

    PLAY
    PAUSE
    TEARDOWN
    GET_PARAMETER
    SET_PARAMETER

3.2 bandwidth Transport Parameter

A client may use the "bandwidth" Transport parameter to reserve bandwidth for a transport. Its argument is a decimal number specifying the bandwidth to reserve in bits per second. If no bandwidth parameter is given, it implies that the media server will use the bit rate of the presentation specified in the SETUP request's URI for the bandwidth of the transport.

The formal syntax for the "bandwidth" parameter is:

```
bandwidth = "bandwidth" "=" 1*DIGIT
```

4 PLAY Queue Enhancements

Requiring all new PLAY requests to be queued when another PLAY request is active makes low-latency implementation of fast forward and rewind difficult; it requires multiple requests to the media server to stop the current PLAY and start the new one. Further, it makes seamless transitions between normal and scaled play impossible, since the current PLAY must be stopped, resulting in a gap in the media delivery, before the new PLAY can be started.

Similarly, requiring all PAUSE requests to flush the queue of PLAY requests is awkward. This forces a client to remember and reissue all previously queued PLAY requests when it restarts a stream after a PAUSE.

These problems can be resolved by allowing clients to specify the type of queuing behavior they desire on each request. The proposed mechanism uses two new headers:

    Play-Now
    No-Flush

4.1 Play-Now Header

A client may use the Play-Now header with either a SETUP or PLAY method.

4.1.1 Play-Now with PLAY

When used in a PLAY request, this header indicates that the PLAY operation should be performed immediately rather than queuing it. Using Play-Now in a PLAY request causes any queued PLAY requests to be discarded unless the No-Flush header is also included.

4.1.2 Play-Now with SETUP

When added to a SETUP request, this header indicates that the client wants streaming to begin immediately (i.e., possibly even before the SETUP response is sent to the client). This allows the client to avoid waiting for the response from SETUP and then issuing a PLAY command, but has some practical limitations.

Play-Now with SETUP is not useful in those environments where the client requires information contained in the SETUP response before it can start decoding the media stream. For example, if a set top box needs the SETUP response to know which channel to tune to, it will typically need to issue a separate PLAY command after it has tuned to the proper channel.

If the Play-Now header is included in a SETUP request, Range and Scale headers may also be included.

4.2 No-Flush Header

A client may use the No-Flush header with either a PAUSE or PLAY method. When added to either request, it prevents queued PLAY requests from being discarded.

4.3 Alternate Approach

An alternate approach to providing the same functionality would be to define a single header with directives along the lines of the Cache-Control header. An example of the syntax is:

```
Queue-Control   = "Queue-Control" ":" queue-directive
                  *(";" queue-directive)
queue-directive = "play-now"
                | "no-flush"
```

5 Server State Changes

Most clients need to track the state of the media server while the server is streaming. The most critical state change to clients occurs when the media server encounters the end of a presentation (or the beginning when rewinding), and stops streaming. There are currently no standard mechanisms for detecting this in the RTSP specification. Problems clients encounter in the current architecture include:

- Polling for the current media server state wastes network bandwidth, and introduces unacceptable latencies in detecting state transitions.

- In non-IP delivery environments, the transport typically remains allocated even if no media is being delivered. This means that a client cannot watch for the server to close the transport to signal the end of media delivery.

- Watching for the incoming media to stop is unreliable. Short timeouts can trigger a false end of media detection if the media flow is temporarily delayed. Long timeouts introduce unacceptable latencies. Clients are unable to distinguish between a normal end of media and an error condition that resulted in the media delivery stopping.

These problems can be remedied by a client callback mechanism. The proposed mechanism uses the ANNOUNCE method sent from the server to the client, along with a new header which contains the details of media server state transitions.

5.1 ANNOUNCE Callbacks

If desired by the client, an ANNOUNCE request can be sent asynchronously from the server to the client to notify it of any changes in a session state. ANNOUNCE requests are only sent to a client if the client used the May-Notify header in its SETUP request for the session (section 5.2). The nature and time of the event causing the stream state change are contained in the Notice header (section 5.3).

An ANNOUNCE request will only be sent if the session is currently associated with an open persistent connection to the client. If the session is not associated with a connection to the client, the state change notification will be returned in the next GET_PARAMETER response for the session.

Alternate approaches would be to use GET_PARAMETER or SET_PARAMETER for callbacks, or to define a new method.

5.2 May-Notify Header

The May-Notify header may be included in a SETUP request.

If a client includes the MayNotify header in a SETUP request, the server will notify the client asynchronously of any stream state changes by sending it an ANNOUNCE request (section 5.1). If this header is not included, state changes are returned to the client as part of a GET_PARAMETER response. In both cases, the state change is reported with a Notice header (section 5.3).

5.3 Notice Header

The Notice header contains media server state change information for a session, such as errors encountered during play or reaching the end of the stream. It may only originate from a media server, and is not recognized in client requests. The Notice header is sent from the server to a client via either an ANNOUNCE request or a GET_PARAMETER response (section 5.1).

The formal syntax for the Notice header is:

```
Notice       = "Notice" ":" notify *("," notify)

notify       = event-code SP """ event-phrase """ SP
               "event-date" "=" utc-time

event-code   = 4DIGIT

event-phrase = *<TEXT, excluding CR, LF, ">
```

Event codes and phrases which may be returned by the server are:

Code	Message
1103	Stream Stalled
1104	Stream Resumed
2101	End-of-Stream Reached
2103	Transition
2104	Start-of-Stream Reached
2306	Continuous Feed Terminated
4401	Error Reading Media Data
5201	Server Resources Unavailable
5401	Stream Failure
5402	Session Terminated by Server
5403	Server Shutting Down
5501	Internal Server Error

6 Miscellaneous

6.1 Reason Header

A client may wish to inform the server why it has chosen to tear down a session. This is often useful in diagnosing server or network problems. This is accomplished with the Reason header. The Reason header is only valid in TEARDOWN requests.

How much of the Reason header message is saved by the media server, or whether the message is saved at all, is up to the discretion of the media server. Implementers of media servers should place limits on the message length and message frequency to prevent the Reason header from being used in denial-of-service attacks.

The formal syntax of the Reason header is:

```
Reason        = "Reason" ":" reason-phrase
reason-phrase = *<TEXT, excluding CR, LF, ">
```

6.2 Looping Ranges

Continuous looping play of a presentation is a frequent requirement in commercial environments. This is typically used for movie trailers, etc.

To support this, the Range header can be enhanced to allow clients to ask the media server to continuously loop a presentation. The formal syntax of the extended Range header is:

```
Range        = "Range" ":" 1\#ranges-specifier *(range-option)
range-option = ";" "time" "=" utc-time
             | ";" "loop" [ "=" loop-count ]
loop-count   = 1*DIGIT
```

Adding the "loop" option to a Range header causes the specified range within the media to loop for "loop-count" iterations, or forever if no "loop-count" is specified.

A PAUSE request or another PLAY request for the session will stop the looping. A PAUSE request will terminate the loop immediately. A queued PLAY request (without the Play-Now header, section 4.1) will terminate the loop at the end of the current iteration. A PLAY request with the Play-Now header will terminate the loop immediately.

6.3 Additional Status Codes

The following two standard status codes should be added:

Code	Message
463	Destination Required
464	Unable to Visual Scan

Code 463 indicates that the media server was unable to select an appropriate transport destination address for the client, and that the client must supply one explicitly. It may only be returned in SETUP responses.

Code 464 may only be returned in response to a PLAY request, which includes a scale other than 1. It indicates that the server is unable to stream the media at a rate other than normal speed forward. This may be a temporary condition caused, for example, by unusually heavy loading on the media server. It may also be a permanent condition due, for example, to media encoding limitations or media server policy.

6.4 Stream Parameters

Standard parameters need to be defined for the GET_PARAMETER method to be generally useful. Proposed standard parameters are:

stream_state	The current stream state. Possible returned values are: playing ready
position	The current stream position. The position is the number of seconds from the beginning of the media in npt format.

Appendix A: Author's Address

Sean Sheedy
nCUBE Corporation
1825 NW 167th Place
Beaverton, OR 97006
USA

E-mail: seans@ncube.com

References

1. Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

2. Handley, M., and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

3. nCUBE Corporation, "nCUBE RTSP Implementation and Extensions", January 2000.

4. Oracle Corporation, "Custom Video Client Developer's Guide, Release 3.2", September 1999.

5. International Telecommunication Union, "Generic Coding of Moving Pictures and Associated Audio Information: Systems", H.222.0, July 1995.

6. International Telecommunication Union, "Multimedia Multiplex and Synchronization for Audiovisual Communication in ATM Environments", H.222.1, March 1996.

7. European Telecommunications Standards Institute, "Digital Video Broadcasting: Framing Structure, Channel Coding and Modulation For Cable Systems", EN 300 429, October 1997.