TechNote – AltitudeCDN™ Multicast+ Integration with LogMeIn GoToWebcast

Version 1.1

AltitudeCDN™ Multicast+ is a patented (US Pat. 9,516,390) solution that brings multicast support to any live video deployment that uses HTTP Live Streaming (HLS) or DASH. Multicast+ integrates with the LogMeIn GoToWebcast video event management service, providing LogMeIn customers with a solution that efficiently delivers high-quality live (or video on demand (VOD) published as live) streaming across the enterprise, without overloading or degrading network resources.

This guide describes how to provision and deploy a GoToWebcast video event that uses AltitudeCDN Multicast+.

Contents

Introduction

During live (or video on demand (VOD) published as live) events, large numbers of viewers attempt to individually connect to external internet content delivery servers, creating a high demand on network resources that can lead to congestion, poor video quality, and even service interruptions for event viewers or other network clients.

To avoid these issues, Ramp and LogMeIn have jointly developed a solution where GoToWebcast customers can integrate AltitudeCDN Multicast+ into their video content management and delivery environment. By using Multicast+ Senders and Receivers, a video event is presented as a single multicast stream that is shared simultaneously by multiple viewers, allowing an event deployment to scale to large audiences, while conserving network bandwidth.

The GoToWebcast event management interface supports parameters to provision Multicast+ events. Typically, LogMeIn configures these parameters for a customer as a managed service, however, the Multicast+-related management interface can be exposed for customer access. The on-premise Multicast+ Sender application must be configured separately to match the corresponding GoToWebcast event, and the Multicast+ Receiver application must be pre-installed for all viewers. The GoToWebcast video player and portal page is enabled to detect and receive Multicast+ streams if present, or fall back to unicast transmission if none are found.

Requirements

To use Multicast+ with GoToWebcast, you need the following:

Item	Recommendation
Multicast+ Sender	Multicast+ Sender v1.6.5 or later – The Sender must be installed and running on an on-premise Windows or CentOS platform, positioned within the customer network so multicast traffic can reach the audience. For more information, see the *AltitudeCDN Multicast+ Deployment Guide*.
Multicast+ Receivers	Any of the following: • Multicast+ Windows Receiver 1.8 or later. • Multicast+ Mac Receiver 1.6.5 or later. The receivers must be installed on each client viewing device prior to the event. For more information, see the *AltitudeCDN Multicast+ Deployment Guide*.

Solution Summary

Event viewers initiate playing the video event normally by accessing the GoToWebcast Player page. The GoToWebcast Player (or any other Multicast+-aware player) then searches for and detects a local Multicast+ Receiver that can receive the pre-configured multicast streams. If no Receiver or multicast stream is available, the GoToWebcast Player falls back to unicast transmission.

The Multicast+ Sender uses a management URI that has been provided for the GoToWebcast event. This URI supplies the Sender Agent with a dynamic configuration that determines which multicast addresses and ports are used for each stream. Any stream switching initiated by the presenter is reflected automatically on the viewer's player.

> ***Note:*** *For information on configuring the Multicast+ Sender, see the AltitudeCDN Multicast+ Deployment Guide.*

To use Multicast+ with GoToWebcast:

1. **(Typically configured by LogMeIn)** Create a matching Multicast+ key pair, then provision:
    - The player landing page for the specific customer.
    - The corresponding senderKeyStore in the customer's on-premise senderAgent.properties file.
2. Configure each GoToWebcast video event stream (for example, Prelive, Primary, and Backup) with a unique multicast address and UDP port pair (for example, 239.1.1.1:3500).

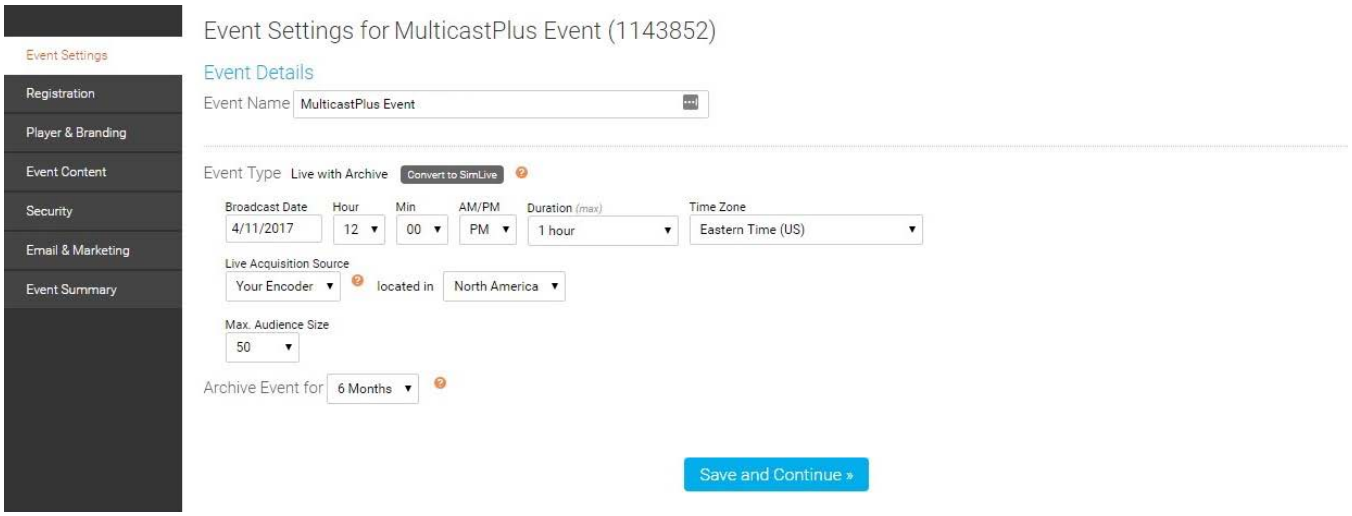
    > ***Note:*** *Multicast address/port pairs for events require unique multicast addresses to prevent overlap issues with other events.*

3. Configure the URI for the local Multicast+ Receiver (for example `http://partners.multicast-receiver-altitudecdn.net:12345`, or `https://partners.multicast-receiver-altitudecdn.net:12350`):
    - HTTPS is recommended to match the scheme of the player portal.
    - partners.multicast-receiver-altitudecdn.net resolves to 127.0.0.1 in the public DNS.
4. LogMeIn provides a source URI that can be added to the senderAgent.properties file. For example:
    - `vdmsagent.managementUri=https://event.webcasts.com/viewer/ramppullpaths.jsp?ei=1143852&sh0=df909e2f4afd61de52eb8e89f3e886c7249c2294`
    - This source URI provides dynamic JSON-based configuration to the Multicast+ Sender agent to support stream switching.

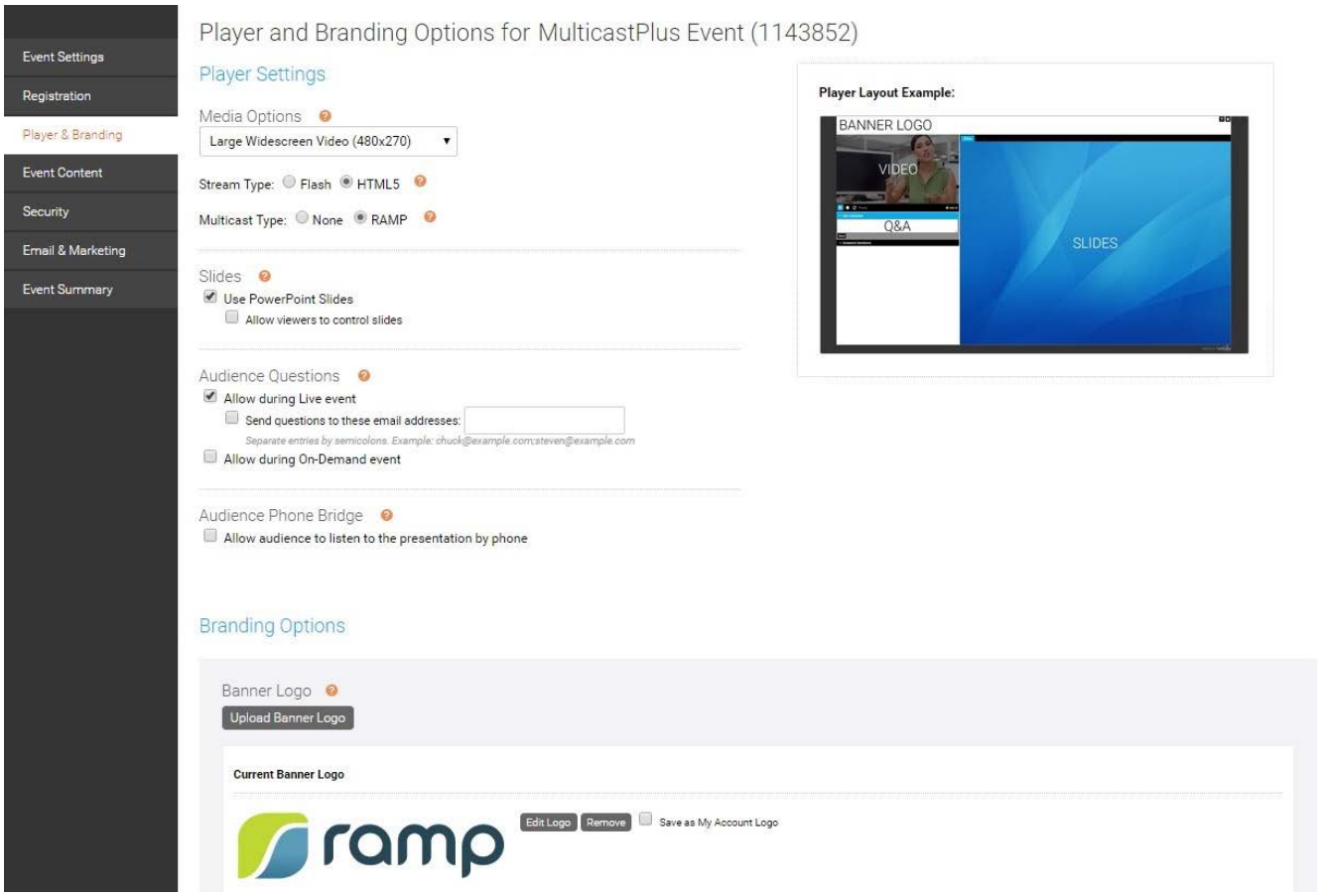
GoToWebcast Multicast+ Configuration

This section summarizes the configuration steps for using Multicast+ with GoToWebcast (for more information on the GoToWebcast interface, see LogMeIn documentation):

1. Create a new Event.
2. Within Event Settings, set the appropriate values for the Event Name, Event Type, and Archive Events settings. Note that only live events are supported by the GoToWebcast Multicast+ integration.



3. Within the Player and Branding Options:
    - For Media Options, select:
        - Stream Type: HTML5
        - Multicast Type: RAMP
    - Set the appropriate values for the Slides, Audience Questions, and Audience Phone Bridge settings.



4. Within Event Summary, display the Ramp Setup panel:
    - Define the Prelive, Primary, and Backup stream multicast IPv4 and ports used for the event. These addresses must be coordinated with the customer network IT to ensure that no collision or overlap occurs with other events or network traffic.
    - A default DNS name (partners.multicast-receiver-altitudecdn.net => 127.0.0.1) is used to identify the local Multicast+ Receiver Host. This hostname can be used for all GoToWebcast deployments.
      A valid PKI certificate for this hostname is embedded in the Multicast+ Receiver to allow HTTPS delivery of video content (for example: `https://partners.multicast-receiver-altitudecdn.net:12350`).

      > ***Note:*** *Ensure that the Receiver Host URL port matches the scheme used (for example, HTTP uses 12345, and HTTPS uses 12350).*

    - Copy the Sender Source URL to the Multicast+ Sender configuration. For example: `vdmsagent.managementUri=https://event.webcasts.com/viewer/ramppullpaths.jsp?ei=1143852&sh0=df909e2f4afd61de52eb8e89fe886c7249c2294`

Multicast+ Sender Agent High-Availability

The Multicast+ Sender Agent can be configured to support high-availability (HA) deployments, where clusters of Multicast+ Sender Agents provide redundancy for multicast channel video transmissions.

To configure high-availability, do the following on each Sender Agent that is a member of the cluster:

1. Download the file "ipaddr.min.js" from the following location and save it to the same directory as the vdms.properties file:

   `https://github.com/whitequark/ipaddr.js/blob/master/ipaddr.min.js`

2. Within the senderAgent.properties file, add the following:

Property	Description
vdmsagent.haCluster	The list of IPv4 addresses of the Sender Agents in the cluster. For example: `vdms.haCluster=<sender1>,<sender2>, ...`
vdmsagent.haPriority	The priority assigned to the current Sender Agent in the cluster. If an issue occurs with the active Sender Agent, the Sender Agent with the next highest priority resumes the transmission. Valid values: Integer, where a lower number denotes a higher priority. For example: vdmsagent.haPriority=*n* ***Note:*** *Priority assignments must be unique to clearly establish precedence.*
vdmsagent.haGroup	Identifies the media server to which the current Sender Agent is connected. For example, if you have two HLS servers, Sender(s) connecting to the first should be configured with haGroup=0 and to the second with haGroup=1. For more information, see the *AltitudeCDN Multicast+ Deployment Guide*. Valid values: 0-9 Default value: 0
vdmsagent. customProcessorJS	A JavaScript processing extension that allows you to define JavaScript code.

	For high-availability, the Example customProcessorJS Function shows how you can use customProcessorJS to create an adapter function that does the following: • Adds custom properties to make each event highly available between the Senders that belong to the cluster. • Select a unicast port based on the lower two octets of the multicast address.

Example customProcessorJS Function

An example of how the customProcessorJS property can be used for HA deployments is provided below:

```
# Start of JS Code
vdmsagent.customProcessorJS=\n\
load("ipaddr.min.js");\n\
var addrexp=new RegExp("([0-9\\.]+)",'g');\n\
function convert(agentProps,stream) {\n\
       var mobj = {\n\
                      "multicastAddress": stream.maddress + "/10",\n\
                      "sourceUrl": stream.url,\n\
                      "customProperties":{}, \n\
                      "D": agentProps.hasOwnProperty("D")? agentProps.D: 20, \n\
                      "sourceFormat": "hls"};\n\
       if (stream.type == "live"){ mobj.sourceType = "live"; }\n\
       else if (stream.type == "od") {\n\
              mobj.sourceType = "vod";\n\
              if (stream.hasOwnProperty("offset")) {\n\
                     mobj.timeOffsetMs = stream.offset * 1000; /* offset in MS */ \n\
              }\n\
       } else {\n\
              /* unhandled stream.type value. */ \n\
              return undefined;\n\
       }\n\
       /* Only active streams are sent.  Negative offset filter: */ \n\
       if (mobj.hasOwnProperty("timeOffsetMs") && mobj.timeOffsetMs < 0){\n\
              return undefined;\n\
       }\n\
       if (mobj.hasOwnProperty("multicastAddress")) {\n\
          var maddr=mobj["multicastAddress"];\n\
          if (maddr.match(addrexp)){\n\
              var parsed=ipaddr.parse(maddr.match(addrexp)[0]);\n\
              /* print("ip: " + parsed.octets); */\n\
              haCluster(agentProps,mobj,parsed);\n\
          }\n\
       }\n\
       return mobj;\n\
};\n\
function haCluster(agentProps, stream, addr) {\n\
  /* print("Called haCluster"); */\n\
  if (stream && addr) {\n\
    /* print("Use HA addr: " + addr); */\n\
    var props = stream.hasOwnProperty("customProperties") ? stream.customProperties : {};\n\
    if (agentProps.hasOwnProperty("vdmsagent.haCluster")) {\n\
      var port = (addr.octets[2]<<8) | addr.octets[3];\n\
      /* print("Use port: " + port); */\n\
      var ha = agentProps["vdmsagent.haCluster"].split(",");\n\
      for (var i = 0; i < ha.length; i++){\n\
        ha[i] = ha[i].trim() + ":" + port;\n\
      }\n\
```

```
      /* print("HA port: " + port + " group: " + ha.join(","));*/\n\
      props["haClusterUnicastAddresses"] = ha.join(",");\n\
      props["haClusterUnicastListenerPort"] = port;\n\
    }\n\
    if (agentProps.hasOwnProperty("vdmsagent.haPriority")) {\n\
      props["haPriority"] = agentProps["vdmsagent.haPriority"];\n\
    }\n\
    if (agentProps.hasOwnProperty("vdmsagent.haGroup")) {\n\
      props["haGroup"] = agentProps["vdmsagent.haGroup"];\n\
    }\n\
    stream["customProperties"]=props;\n\
  }\n\
};\n\
function addChannels(agentProps,incoming,req) {\n\
       if (incoming.hasOwnProperty("streams")) {\n\
              var allChan = {};\n\
              for (var property in incoming.streams) {\n\
                     if (incoming.streams.hasOwnProperty(property)){\n\
                            var stream = incoming.streams[property];\n\
                            var mobj = convert(agentProps,stream);\n\
                            if (mobj !== undefined) {\n\
                                   allChan[stream.streamid] = mobj;\n\
                            }\n\
                     }\n\
              }\n\
              if (Object.keys(allChan).length > 0) { req.channels = allChan; };\n\
       }\n\
};\n\
function initRequest(agentProps, orig){\n\
  var req = {};\n\
  req.senderKeyStore = agentProps.senderKeyStore;\n\
  if (orig.hasOwnProperty("senderKeyStore")) {\n\
    req.senderKeyStore = agentProps.senderKeyStore;\n\
  }\n\
  return req;\n\
};\n\
function request(body,bodyStr,agentProps,env) {\n\
   if (bodyStr){\n\
     var incoming;\n\
     try {\n\
      incoming = JSON.parse(bodyStr);\n\
     } catch(e) {\n\
      print(bodyStr);\n\
      return {};\n\
     }\n\
     var outgoing = initRequest(agentProps, incoming);\n\
     /*\n\
     print ("-----------------------------------------------------");\n\
     print("INIT: " + JSON.stringify(outgoing, null, '    '));\n\
     print("INCOMING REQUEST: " + JSON.stringify(incoming, null, '    '));\n\
     print("AgentProperties: " + JSON.stringify(agentProps, null, '    '));\n\
     */\n\
     addChannels(agentProps,incoming,outgoing);\n\
     /*\n\
     print("OUTGOING REQUEST: " + JSON.stringify(outgoing, null, '     '));\n\
     print ("-----------------------------------------------------");\n\
     */\n\
     return outgoing;\n\
   }\n\
   return {};\n\
}
#  End of JS Code
```

Bandwidth-based HLS Stream Variant Selection

The Multicast+ Sender (version 1.8.2 and later) allows you to control the selection of HLS stream variants for transmission by using the “hlsIgnoreResolutionInDeterminingMaxBandwidth” property.

HLS stream variants, such as the un-transcoded source stream from Wowza encoders, can be included in a manifest without specifying a resolution, and consequently, cannot be selected for transmission over other streams that include a resolution.

When the Multicast+ Sender is configured with “hlsIgnoreResolutionInDeterminingMaxBandwidth=true”, the stream with the highest bandwidth (typically the un-transcoded source stream) is selected, rather than the stream with the highest resolution.

To enable bandwidth-based HLS stream variant selection, do either of the following:

- If your Multicast+ deployment uses sender.bat/sender.sh to start the Multicast+ Sender, add the “hlsIgnoreResolutionInDeterminingMaxBandwidth=true” property to the sender.properties file.
- If your Multicast+ deployment uses senderAgent.bat/senderAgent.sh to start the Multicast+ Sender, add the “vdmsagent.hlsIgnoreResolutionInDeterminingMaxBandwidth=true” property to the agent.properties file using the customProcessorJS function as described above.

Diagnostic Usage Statistics

To enable Multicast+ Sender diagnostics, you can run an additional instance of a Sender on the same platform as the Sender Agent by doing the following:

1. Add the following minimal configuration to the Diagnostics Server’s properties file:
```
# port both to receive data and serve http
diagnostics.port=18000
diagnostics.regionsFile=regions.json
diagnostics.password=abc123
diagnostics.noAuth
```
2. Start the Diagnostics Server using the following command:
```
# nohup ./sender.sh diagnostics.properties > /dev/null 2>&1 &
```