SIP Endpoint SDK Developer's Guide

Audio statistics and MOS Calculation

12/14/2025

Contents

Audio statistics and MOS Calculation

This page describes the API to get audio statistics for current or recently completed call, what is included into that statistics, and how the derived metrics (such as MOS) are calculated.

Internal C++ data structure

```
typedef struct gs_call_statistics {
  int got_local_stat; // true if next 5 fields are valid (local stat available)
  float local_fraction_lost; // fraction of lost packets (not locally received)
  int      local_total_lost; // total lost packaged (calculated per RFC 3550)
  unsigned local_jitter;     // interarrival jitter
  unsigned local_PktCount;   // local packet count (number of RTP packets sent)
  unsigned local_OctCount;   // local octet count (number of bytes sent)
  int got_remote_SR; // true if next 2 fields valid (got remote Sender Report)
  unsigned remote_PktCount;  // remote packet count
  unsigned remote_OctCount;  // remote octet count
  int got_remote_RR; // true if next 3 fields valid (got remote Receiver Report)
  float remote_fraction_lost; // fraction of lost pkts (not received by remote)
  int      remote_total_lost; // total lost packaged (calculated per RFC 3550)
  unsigned remote_jitter;     // interarrival jitter
  int rttMs; // Round-trip time in milliseconds.
  float local_MOS;      // local MOS for entire call segment (added in 9.0.005)
  float local_interMOS; // local MOS for the last interval
  //
  // added in 9.0.009 (all decoded_10ms_xxx values are numbers of 10ms frames):
  //
  int got_neteq_stat; // true if next 6 fields are valid (NetEQ stat available)
  int     local_pkt_received; // total number of RTP packets received locally
  pgtime_t last_pkt_received; // timestamp of last RTP or RTCP packet received
  int decoded_10ms_normal;    // AudioDecodingStats - normal (decoded from RTP)
  int decoded_10ms_plc;       // - Packet Loss Concealment
  int decoded_10ms_cng;       // - Comfort Noise Generation
  int decoded_10ms_silence;   // - dead silence
  //
  // information about current (or last used) codec, added in 9.0.012:
  //
  int got_codec_info; // true if next 2 fields valid (codec info is available)
  int         codec_rtp_pt;   // payload type used for sending RTP packets
  const char *codec_sdp_name; // codec name (as appears in SDP)
}
gs_call_statistics;
```

OS X

The following methods are defined in GSSessionService protocol to get audio statistics:

```
  @protocol GSSessionService <NSObject>
  /**
   Get audio statistics for particular session
   @param session object.
   @returns GSStatistics with audio statistics content.
```

```
 */
- (GSStatistics*) audioStatisticsForSession:(id<GSSession>) session;
- (GSStatistics*) audioStatisticsForSessionId:(int) sessionId;
```

where GSStatistics is an object with the following properties (closely matching internal C++ structure, just with names formatted using standard objective-C convention):

```
@property (nonatomic) int gotLocalStat;
@property (nonatomic) float localFractionLost;
@property (nonatomic) int localTotalLost;
@property (nonatomic) unsigned localJitter;
@property (nonatomic) unsigned localPktCount;
@property (nonatomic) unsigned localOctCount;
@property (nonatomic) int gotRemoteSR;
@property (nonatomic) unsigned remotePktCount;
@property (nonatomic) unsigned remoteOctCount;
@property (nonatomic) int gotRemoteRR;
@property (nonatomic) float remoteFractionLost;
@property (nonatomic) int remoteTotalLost;
@property (nonatomic) unsigned remoteJitter;
@property (nonatomic) int rttMs;
@property (nonatomic) float vqLocalMOS;
@property (nonatomic) float vqLocalIntervalMOS;
@property (nonatomic) int gotNeteqStat;
@property (nonatomic) int localPktReceived;
@property (nonatomic) struct timeval lastPktReceived;
@property (nonatomic) int decoded10msNormal;
@property (nonatomic) int decoded10msPlc;
@property (nonatomic) int decoded10msCng;
@property (nonatomic) int decoded10msSilence;
```

.NET

The following method is defined in ICallControl interface to get audio statistics for specific session ID:

```
Dictionary<String^,Object^>^ GetAudioStatistics(int sessionId);
```

where the returned dictionary is created from internal gs_call_statistics structure as following:

```
virtual Dictionary<String^, Object^>^ GetAudioStatistics(int sessionId)
{
  Dictionary<String^,Object^>^ res = gcnew Dictionary<String^,Object^>();
  gs_call_statistics gst = this->endpoint->getAudioStatistics(sessionId);
  res->Add("Got Local Stat ", gst.got_local_stat);
  res->Add("Local Frac Lost", gst.local_fraction_lost);
  res->Add("Local TotalLost", gst.local_total_lost);
  res->Add("Local Jitter   ", gst.local_jitter);
  res->Add("Local Pkt Count", gst.local_PktCount);
  res->Add("Local Oct Count", gst.local_OctCount);
  res->Add("Got Remote SR  ", gst.got_remote_SR);
  res->Add("Remote PktCount", gst.remote_PktCount);
  res->Add("Remote OctCount", gst.remote_OctCount);
  res->Add("Got Remote RR  ", gst.got_remote_RR);
  res->Add("Remote FracLost", gst.remote_fraction_lost);
  res->Add("Remote TotaLost", gst.remote_total_lost);
  res->Add("Remote Jitter  ", gst.remote_jitter);
  res->Add("Round Trip Time",      gst.rttMs);
  res->Add("VQ Local MOS",         gst.local_MOS);
```

```
    res->Add("VQ Local IntervalMOS", gst.local_interMOS);
    res->Add("GotNeteqStat",       gst.got_neteq_stat);
    res->Add("LocalPktReceived",   gst.local_pkt_received);
    res->Add("LastPktReceived",    gst.last_pkt_received);
    res->Add("Decoded10msNormal",  gst.decoded_10ms_normal);
    res->Add("Decoded10msPlc",     gst.decoded_10ms_plc);
    res->Add("Decoded10msCng",     gst.decoded_10ms_cng);
    res->Add("Decoded10msSilence", gst.decoded_10ms_silence);
    return res;
  }
```

Detailed Description

Data returned by SIP Endpoint SDK includes the following raw audio statistics data and derived metrics (refer to following sub-sections for detailed description):

- raw local stats (used to generating RTCP reports),
- remote stats obtained from RTCP reports from other side of conversation,
- estimated MOS values, calculated based on the raw statistics,
- audio decoding statistics from Voice Engine
- current codec info

All locally gathered / calculated statistics are reset each time when RTP session is started (after hold/retrieve or because of codec changed for active session). The SDK always returns statistics for last call segment only, from the moment last RTP session for the call was started.

Raw local stats

This group includes raw local stats that are used for RTCP reports generation, with their meaning defined by RFC 3550, namely:

- `float local_fraction_lost` = same as "fraction lost" in RFC 3550, but expressed as float number
- `int local_total_lost` = "cumulative number of packets lost" in that RFC
- `unsigned local_jitter` = "interarrival jitter"
- `unsigned local_PktCount` = "sender's packet count"
- `unsigned local_OctCount` = "sender's octet count" (for this Endpoint considered a "sender")

These values are always available as soon as RTP transmission starts, right after SDP negotiation is completed and audio devices initialized.

Remote stats from RTCP reports

These two groups include remote-side stats received in RTCP reports, with the same meaning as local stats described in previous section. This information is available only after receiving RTCP packet with Server Report (SR) for remote_Pkt/OctCount values, or Receiver Report (RR) for the other 3 values (for successfully established call, usually RTCP packet contains both reports).

While RFC 3550 provides quite complex rules and algorithms for calculating RTCP transmission interval, most modern implementations (including Geneys SIP Endpoint SDK) use simplified algorithm, sending RTCP with random intervals between 2.5 and 7.5 sec (i.e. on average one RTCP every 5 sec). Since first RCTP packet may not include all data yet, full remote stats are likely to be available only ~10 sec after the media session is started.

Calculated RTT and MOS

This group includes:

- round-trip time (RTT) calculated based on timing date in received RTCP packets, value in milliseconds,
- estimated Mean Opinion Score (MOS) calculated using "E model" from ITU-T recommendation G.107 (see details below), with two value provided:
  - `local_MOS` calculated over the entire duration of RTP session (from the time call was first established, retrieved from hold, or codec was changed),
  - `local_interMOS` calculated for the last RTCP transmission interval (value recalculated every time RTCP is sent)

For all 3 statistics, zero value means "data is not available", i.e. no RTCP received yet to calculate RTT, or not enough incoming RTP packets received to estimate the MOS (current implementation requires at least 5 incoming RTP to start calculations). Refer to the separate section below for the details of MOS calculation.

Audio decoding statistic

This group includes audio processing statistics from Voice Engine network equalizer / decoder:

- total number of RTP packets received locally,
- timestamp of last RTP or RTCP packet received,
- number of 10-msec audio frames that were decoded as:
  - "normal" (i.e. fully from the RTP packet received),
  - affected by Packet Loss Concealment (PLC),
  - added by Comfort Noise Generation (CNG) - only present when remote side uses silence suppression along with Comfort Noise (CN) packets,
  - inserted dead silence.

Because of inevitable delays in network transmission (and two sides not being able to start media stream at exactly the same time), small number of PLC and "silence" frames are fully expected even in perfect conditions.

Codec info

This group includes current (or last used, for inactive session) codec information:

- codec_rtp_pt is RTP payload type as defined in RFC 3550. For most commonly used codec that is a static values assigned by RFC 3551 (that parameter is not very useful for codecs with dynamic payload type):

  - for G.711 - pt=0,"pcmu" or pt=8,"pcma"
  - for G.722 - pt=9,"g722"
  - for G.729 - pt=18,"g729"
- codec_sdp_name is canonical codec name as it appears in SDP, e.g. "opus", "iSAC" or "iLBC" for the rest of audio codecs

MOS calculation in SIP Endpoint SDK

Currently SIP Endpoint SDK uses "E model" from ITU-T recommendation G.107 for MOS calculation, with codec-specific Transmission impairment factors from recommendation G.113. Namely, the R-factors (for Listening and Conversation Quality) are calculated as following:

```
R(LQ) = R0 - Is - Ie-eff + A,  and R(LQ) = R(CQ) - Id
where:
  R0 is Basic Signal to Noise Ratio | narrowband: R0-Is = 93.25
  Is is Simultaneous Impairments    | wideband:   R0-Is = 120
   A is Advantage Factor = 0 constant

Id = Idte + Idle + Idd,  where: Idte is Talker Echo = 0 constant
                                Idle is Listener Echo = 0.15 constant

Idd = If Ta <= 100 ms, Idd = 0     {           1/6               1/6   }
                                   { [       6 ]      [         6 ]     }
      If Ta  > 100 ms, Idd =  25 x { [ 1 + [Y] ]  - 3x[ 1 + [Y/3] ] + 2 }

             log(Ta/100)
  where: Y = -----------        (Ta is estimates as half round-trip time)
                log2
                           Ppl
Ie-eff = Ie + (95 - Ie) x -------
                        Ppl + Bpl
where:
  Ie is Codec-based Equipment Impairment Factor defined in ITU-T G.113
  Ppl is Packet Loss %
  Bpl is Codec-based Packet Loss Robustness Factor defined in ITU-T G.113
```

After that, MOS is calculated from R(LQ) as following:

```
Rx = codec_is_wideband ? R/1.29 : R;
MOS = If Rx < 0;        MOS = 1             [7Rx(Rx - 60)(100 - Rx)]
      If 0 <= Rx < 100; MOS = 1 + 0.035Rx + ------------------------
      If Rx >= 100;     MOS = 4.5                     10^6
```

Codec-based impairment and robustness factors are listed in the following table, along with maximum possible R(LQ) and MOS values, which can be achieved in "perfect" conditions, i.e. with no packet loss, low jitter (completely covered by jitter buffer) and small round-trip time (below 100 ms):

Codec	Ie	Bpl	max_R	max_MOS
G.711 (pcmu or pcma)	0	10.0	93	4.4
G.722	5	7.1	88	4.3
G.729	10	19.0	83	4.1

Codec	Ie	Bpl	max_R	max_MOS
Opus	16	17.0	93	4.4
iLBC	10	28.0	83	4.1
iSAC (wideband)	0	10.0	120	4.4

The third-party VoIP Quality and Bandwidth Calculator may be used for calculating expected MOS for different packet loss rates.