We added some logs in the firefox soure code (version: 42) and built it.

It’s the modification below:

**File** : Rtp\_sender\_video.cc

**Function**: bool RTPSenderVideo::Send

**Code**:

bool RTPSenderVideo::Send(const RtpVideoCodecTypes videoType,

 const FrameType frameType,

 const int8\_t payloadType,

 const uint32\_t captureTimeStamp,

 int64\_t capture\_time\_ms,

 const uint8\_t\* payloadData,

 const uint32\_t payloadSize,

 const RTPFragmentationHeader\* fragmentation,

 const RTPVideoTypeHeader\* rtpTypeHdr) {

 uint16\_t rtp\_header\_length = \_rtpSender.RTPHeaderLength();

 int32\_t payload\_bytes\_to\_send = payloadSize;

 const uint8\_t\* data = payloadData;

 size\_t max\_payload\_length = \_rtpSender.MaxDataPayloadLength();

 scoped\_ptr<RtpPacketizer> packetizer(RtpPacketizer::Create(

 videoType, max\_payload\_length, rtpTypeHdr, frameType));

 // TODO(changbin): we currently don't support to configure the codec to

 // output multiple partitions for VP8. Should remove below check after the

 // issue is fixed.

 const RTPFragmentationHeader\* frag =

 (videoType == kRtpVideoVp8 || videoType == kRtpVideoVp9) ? NULL : fragmentation;

 packetizer->SetPayloadData(data, payload\_bytes\_to\_send, frag);

 bool last = false;

 while (!last) {

 uint8\_t dataBuffer[IP\_PACKET\_SIZE] = {0};

 size\_t payload\_bytes\_in\_packet = 0;

 if (!packetizer->NextPacket(

 &dataBuffer[rtp\_header\_length], &payload\_bytes\_in\_packet, &last)) {

 LOG(LS\_WARNING) << "fail to get next package!";

 return false;

 }

 // Write RTP header.

 // Set marker bit true if this is the last packet in frame.

 \_rtpSender.BuildRTPheader(

 dataBuffer, payloadType, last, captureTimeStamp, capture\_time\_ms);

 if (SendVideoPacket(dataBuffer,

 payload\_bytes\_in\_packet,

 rtp\_header\_length,

 captureTimeStamp,

 capture\_time\_ms,

 packetizer->GetStorageType(\_retransmissionSettings),

 packetizer->GetProtectionType() == kProtectedPacket)) {

 LOG(LS\_WARNING) << packetizer->ToString()

 << " failed to send packet number "

 << \_rtpSender.SequenceNumber();

 }

 LOG(LS\_INFO) << packetizer->ToString() << " succeed to send packet number: " << \_rtpSender.SequenceNumber()

 << " timestamp: " << \_rtpSender.Timestamp()

 << " SSRC: " << \_rtpSender.SSRC();

 }

 TRACE\_EVENT\_ASYNC\_END1(

 "webrtc", "Video", capture\_time\_ms, "timestamp", \_rtpSender.Timestamp());

 return true;

}

**File:**rtp\_receiver\_video.cc

Function: int32\_t RTPReceiverVideo::ParseRtpPacket

Code：

int32\_t RTPReceiverVideo::ParseRtpPacket(WebRtcRTPHeader\* rtp\_header,

 const PayloadUnion& specific\_payload,

 bool is\_red,

 const uint8\_t\* payload,

 uint16\_t payload\_length,

 int64\_t timestamp\_ms,

 bool is\_first\_packet) {

 TRACE\_EVENT2("webrtc\_rtp",

 "Video::ParseRtp",

 "seqnum",

 rtp\_header->header.sequenceNumber,

 "timestamp",

 rtp\_header->header.timestamp);

 LOG(LS\_INFO) << "Video Receive: seqnum : " << rtp\_header->header.sequenceNumber << " timstamp: " << rtp\_header->header.timestamp;

 rtp\_header->type.Video.codec = specific\_payload.Video.videoCodecType;

 const uint16\_t payload\_data\_length =

 payload\_length - rtp\_header->header.paddingLength;

 if (payload == NULL || payload\_data\_length == 0) {

 return data\_callback\_->OnReceivedPayloadData(NULL, 0, rtp\_header) == 0 ? 0

 : -1;

 }

 // We are not allowed to hold a critical section when calling below functions.

 scoped\_ptr<RtpDepacketizer> depacketizer(

 RtpDepacketizer::Create(rtp\_header->type.Video.codec));

 if (depacketizer.get() == NULL) {

 LOG(LS\_ERROR) << "Failed to create depacketizer.";

 return -1;

 }

 rtp\_header->type.Video.isFirstPacket = is\_first\_packet;

 RtpDepacketizer::ParsedPayload parsed\_payload;

 if (!depacketizer->Parse(&parsed\_payload, payload, payload\_data\_length))

 return -1;

 rtp\_header->frameType = parsed\_payload.frame\_type;

 rtp\_header->type = parsed\_payload.type;

 return data\_callback\_->OnReceivedPayloadData(parsed\_payload.payload,

 parsed\_payload.payload\_length,

 rtp\_header) == 0

 ? 0

 : -1;

}

Result:

Send:

Receive:

