**3GPP TSG-SA WG4 Meeting #130S4-241837**

**USA, Orlando, 18 – 22 November 2024**

**Source: Huawei, HiSilicon**

**Title: Update to Solution #13 for KI #12 adding statistics and some editorial improvement**

**Spec: 3GPP TR 26.822 version 1.0.1**

**Agenda item: 10.6**

**Document for: Agreement**

**1. Introduction**

In solution #13 in TR 26.822 clause 6.13 experiments are described involving RTP Senders, the following 3 implementations are tested.

* Hangout/WebRTC case in a browser (edge/chrome) on windows/Linux PC
* Sender and receiver implementation on Windows/Linux PC
* Server implementations such as MTX media server and Wowza streaming server

The current experimental results illustrate graphs showing the corresponding packet data traffic.

This paper presents additional statistics and the methodology of collecting the statistics.

In Real Time Communication the communication is usually duplex.

For the burst statistics traffic from host A to Host B (a) and Host B to Host A (b) are considered separately (in some cases only one way is considered when there is no duplex communication

Therefore each packet trace will need to be analyzed twice, once for traffic from A to B (a) and vice versa (b) from B to A.

NOTE: Additional hosts can take part in conference calls but in this study only 2 hosts were considered so far.

NOTE: In some cases the return channel has traffic (duplex scenarios), in case of unidirectional flow there is no need to monitor and report traffic

The following statistics for dynamic traffic characteristics are retrieved from the packet traces.

Table 1 Metrics and statistics considered for dynamic traffic characteristics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Metric** | **meaning** | **Additional notes** | **Statistics** |
| Bt | Burst Time (Bt) | The time of the burst (corresponding to the first packet), measured in seconds | Bursts are measured as instantaneous transmission of 2 or more packets | Deterministic |
| Ibt | Inter Burst Time (Ibt) | The average duration between 2 consecutive bursts (burst), measured in seconds | This does not consider single ‘lone’ packets transmitted between the bursts | Average and Standard Deviation |
| Bd | Burst Duration (Bd) | The duration between start and end of a burst, measured in seconds | Measures the duration of the burst (in a maximum timeslot of 2 milliseconds ) | Average and Standard Deviation |
| Bs | Burst Size (Bs) | The Size of the Data Burst measured in bytes | Measures the size of the entire burst (in a maximum timeslot of 2 milliseconds) | Average and Standard Deviation |
| Ppb | Packets per burst (Ppb) | Number of packets in each burst | Only bursts of 2 packets or more are considered | Average and Standard deviation |
| #bp | Total Burst packets (#bp) | Number of packets in a burst | This is the number of packets in the trace that is part of a burst | Deterministic |
| #lp | Total Lone packets (#lp) | Number of packets not in a burst | This is the number of packets in the trace that is part not part of a burst | Deterministic |
| #bb | Total Burst Bytes (#bb) | Number of bytes that are part of a burst | This is the number of bytes that is part of a burst | Deterministic |
| #lb | Total Lone Bytes (#lb) | Number of bytes that are not part of a burst | This is the number of bytes in lone packets | Deterministic |
| %lp | Percentage of lone packets (%lp) | Ratio of lone packets to all packets |  | Percentage |
| %bp | Percentage of burst packets (%bp) | Ratio of burst packets to all packets |  | Percentage |
| %lb | Percentage of lone bytes (%lb) | Ratio of lone packet bytes to all bytes |  | Percentage |
| %bb | Percentage of burst bytes (%bb) | Ratio of burst packet bytes to all bytes |  | Percentage |

For brevity we leave out the deterministic metrics bt, tbs, #bp, #lp, #bb and #lb that depend heavily on the length of the trace. Instead we focus on the ratio between lone packets/bytes and burst packets/bytes and other metrics that are independent of the length of the trace.

Table 2 Scenario descriptions and coding in solution #13

|  |  |
| --- | --- |
| Scenario code |  |
| w\_h | Windows hangout without network throttling duplex case (Figure 6.13.2.2.2-1) [1] |
| u\_h | Ubuntu Hangout with network throttling simplex case (Figure 6.13.2.2.2-2)[1] |
| g\_v\_a | Gstreamer video only measured on machine A (Figure 6.13.2.3.2-2) [1] |
| g\_v\_b | Gstreamer video only measured on machine B (Figure 6.13.2.3.2-3) [1] |
| g\_va\_a | Gstreamer pcm audio + video at receiver (Figure 6.13.2.3.3-1) [1] |
| g\_va\_b | Gstreamer pcm audio + video at sender (Figure 6.13.2.3.3-2)[1] |
| w\_s | Wowza at server measured at server (Figure 6.13.2.4.2-2) [1] |
| w\_r | Wowza stream measured at player receiver (Figure 6.13.2.4.2-3) [1] |
| m\_rv | MTX server measured vlc player receiver (Figure 6.13.2.4.2-5) [1] |
| m\_sav | MTX server video + PCM (Figure 6.13.2.4.2-7) [1] monitored at server |
| m\_rav | MTX server video + PCM (Figure 6.13.2.4.2-7) [1] monitored at vlc client |

Table 3 provides the statistical results for each of the experiments in different scenarios.

Table 3 Statistical results of experiments in different scenarios in scenario #13

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Scenario  (direction) | ibt  (mean) | Ibt  (stdev) | Bd  (mean) | Bs  (mean) | Ppb  (mean) | %lp  (%) | %bp  (%) | %lb  (%) | %bb  (%) |
| w\_h (a) | 0.0352 | 0.01437 | 0.0002 | 3463 | 3.55 | 23.2 | 76.7 | 6.6 | 93.4 |
| w\_h (b) | 0.01592 | 0.019 | 0.00016 | 3756 | 3.3 | 19.2 | 80.8 | 17.4 | 82.6 |
| u\_h (a) | 0.03132 | 0.00873 | 0.00017 | 13895 | 12,3 | 40.2 | 59.8 | 22.6 | 77.4 |
| u\_h (b) | 0.42321 | 0.37918 | 0.00007 | 213.92 | 2 | 72.2 | 27.8 | 73.8 | 26.2 |
| g\_v\_a | 0.01642 | 0.01175 | 0.00052 | 2864 | 7.73 | 2.4 | 97.6 | 0.7 | 99.3 |
| g\_v\_b | 0.01796 | 0.00967 | 0.00029 | 3109 | 8.6 | 0.2 | 99.8 | 0.2 | 99.8 |
| g\_va\_a | 0.01252 | 0.0114 | 0.00071 | 4144 | 6.6 | 14.2 | 85.8 | 20.1 | 79.9 |
| g\_va\_b | 0.01414 | 0.00955 | 0.00033 | 4472 | 7.5 | 14.8 | 85.2 | 24.7 | 75.3 |
| w\_s (a) | 0.08732 | 0.03483 | 0.00079 | 24822 | 42.3 | 0.2 | 99.8 | 0.0 | 100.0 |
| w\_r (a) | 0.03778 | 0.04627 | 0.00054 | 10671 | 18.24 | 0.7 | 99.3 | 0.1 | 99.9 |
| m\_rv | 0.02749 | 0.0404 | 0.00084 | 6630 | 6.38 | 2.5 | 97.5 | 0.5 | 99.5 |
| m\_sav (a) | 0.04777 | 0.03838 | 0.00056 | 11906 | 12.18 | 12.6 | 87.4 | 3.3 | 96.7 |
| m\_rav (a) | 0.02573 | 0.03412 | 0.00086 | 6692 | 6.8 | 12.2 | 87.8 | 3.2 | 96.5 |

Some observations of the statistics

* A large percentage of the traffic and packets is part of a burst, highlighting the relevance of burst handling
* Burst durations are shorter than a millisecond
* Results do not say anything on predictability, this is implementation dependent
* PCM audio is tested, transmitted in lone packets, this increases the share of lone bytes ( compared to more efficiently compressed audio)

**2. Reason for Change**

Add the statistics and aggregate results to the TR 26.822 solution #13 as they provide useful information about the experiment.

In addition some editorial improvements to the text are included in this Pseudo CR.

**3. Conclusions**

The tables and statistics provide useful aggregate data on dynamic data bursts for each of the different scenarios that were tested.

**4. Proposal**

It is proposed to agree the following changes to 3GPP TR 26.822

|  |
| --- |
| CHANGE 1 |

# References

........

[ZZ] [Real-Time Transport Protocol (RTP) Parameters (iana.org)](https://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml)

[ZZa] 3GPP TS 29.571 3rd Generation partnership project; technical specification group core network and terminals; 5G System; common data types for Service based interfaces; stage 3

[ZZb] 3GPP TS 38.415 NG-RAN; PDU Session User Plane Protocol (Release 18)

|  |
| --- |
| CHANGE 1 |

## 6.13 Solution #13: Dynamic traffic characteristics of RTP sender implementations

### 6.13.1 Key Issue mapping

This is a solution to KI #12 Enhancements of Data Burst Marking.

### 6.13.2 Description

#### 6.13.2.1 General

A data burst indicates a set of multiple PDUs generated and sent in a short period of time as defined in clause 3.1 of TS 23.501 [3]. Data burst is a common dynamic traffic characteristic in communication networks.

The source of traffic in this study is the Real Time Transport Protocol (RTP) as used for conversational or Extended Reality applications and conversational applications. Most experiments use traffic originating from a camera (conversational application), but also some synthetic sources are used that may be more suitable for the case of extended reality traffic.

The data burst identification could be beneficial for the 5GS network, e.g., power saving and efficient radio resource management. In Release 18, the End of Data Burst indication has been introduced to enable the UE power saving in the 5GS, i.e., the NG-RAN node can configure to move a UE into CDRX for power saving after transmitting the end PDU of the data burst. In Release 19, the data burst size has been concluded to enable the RAN radio resource management as described in clause 8.5 of TR 23.700-70 [6].

This solution aims to observe dynamic traffic characteristics from popular RTP senders and possible implications for low latency and real-time communication.

In this solution we explore dynamic traffic characteristics of the following implementations:

- The open source WebRTC implementation of an RTP Sender in a browser implementation.

- The cross platform GStreamer [39] implementation of RTP senders for peer-to-peer (P2P) transmission.

- The sending and receiving measured of two popular real-time streaming servers.

#### 6.13.2.2 WebRTC and paced sender implementation

##### 6.13.2.2.1 WebRTC and paced sender implementation

WebRTC (Web Real-Time Communication) [28] is an open-source project providing web browsers and mobile applications with real-time communication (RTC) via application programming interfaces (APIs). It allows audio and video communication and streaming in web pages by direct peer-to-peer communication, eliminating the need to install plugins.

It can be implemented in web browsers and is a likely source of real-time conversational RTP traffic in practice.

Based on the documentation available [28] from November 2021 in the WebRTC source code this clause describes the way packets may be generated and transmitted in a WebRTC environment.

WebRTC does not send out data packets generated by an encoder directly, instead a pacing module is used that limits and smoothens the packet train that is transmitted out to the network.

The idea is that, say for example, a 30 fps stream of 6 MBps is sent to the network, in the ideal case this would result in equally sized frames of around 25 kiloBytes in around 21 equally sized packets. While in practice, over a 1 second sliding window the average bit-rate of 6 Mbps might be measured, on a shorter timescale there may be periodic bursts that overshoot this average bit-rate with a much larger instantaneous bit-rate. One of the causes could be a sudden movement in a frame.

To avoid potential impact of these cases on the network and application, the WebRTC solution introduces the paced sender. The paced sender introduces a buffer in which the media packets are queued before being sent out. This avoids, for example, that large video frames are split into a large burst of packets that will disrupt other streams such as audio streams that are critical for intelligibility. WebRTC uses multiplexed RTP transport with different streams sent on the same port and IP tuple, thus potentially such an overshoot in video packets could stop other packets from being sent out in time as the socket may be blocked when transmitting the video packets.

NOTE: The standard approach in RTP is to use different ports for different media streams, but in practice this could have some drawbacks as this increases resource usage and may complicate NAT traversal. In this case, the prioritization would depend on the operating system implementation. WebRTC allows multiplexing audio and video on the same port by establishing a relationship between the media lines included in the SDP using the attribute a=group:BUNDLE.

For sending out the media, a leaky bucket algorithm is used for pacing them on to the network. In this implementation packets are inserted into a buffer first before being transmitted. The buffer contains separate first in first out (FIFO) queues for each media type/track, such that audio can for example be prioritized over video. Equal priority streams can be sent in round-robin fashion, avoiding these streams from blocking each another.

In a leaky bucket, the rate of packets that is sent out is limited, potentially leading to a buffering of the input in case the input is larger than the output rate. A leaky bucket is one way to limit the burst sizes in traffic sent to the network.

In WebRTC this output rate is called the *pacing\_rate.*

The typical way a packet is generated and paced on to the network in WebRTC is as follows:

1. Media frame is packetized into RTP packets

2. The packets are sent for scheduling (transmission)

3. The pacer is called to enqueue the packet

4. The packets stay in the queue until the pacer finds a scheduled moment to transmit them out based on the leaky bucket algorithm

5. At the time that the packet is scheduled, the packet is forwarded to the RTP module for final timestamping

6. The packet is sent on the low level interface (i.e. the UDP socket), and it is now out of scope of the WebRTC implementation

Asynchronous to this, the WebRTC implementation tries to estimate the sender bandwidth in order to set the *pacing\_rate,* i.e. the rate at which packets are sent to the network and secondly the *padding\_rate*, the bit rate (if any) used to send padding in case no packets are available to be sent out*.* Figure 6.13.2.2.1-1 illustrates the WebRTC transmission flow.

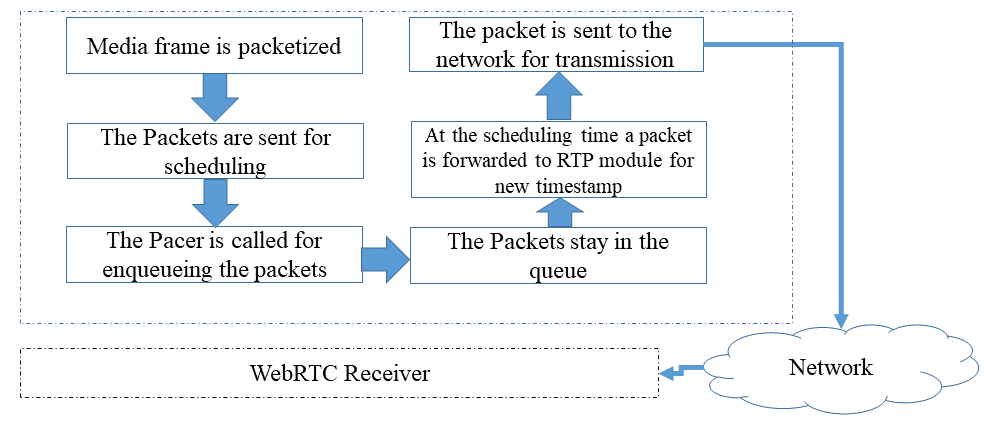


Figure 6.13.2.2.1-1: Packet processing in WebRTC paced sender of an RTP packet

The pacer of WebRTC prioritizes based on different criteria:

a) Packet Type, with most to least prioritized:

1. Audio

2. Retransmission

3. Video and FEC

4. Padding

b) Based on the enqueuing order.

When the queues are empty the implementation will aim to generate padding frames and send them out at the *padding\_rate*. In some cases the *pacer\_rate* may be overridden and ignored, when there is a significant encoding overshoot for example. In addition WebRTC implementation enables setting the maximum time a packet can stay in a queue using a function (*setQueueTimeLimit(TimeDelta limit))*.

Bandwidth estimation in the WebRTC implementation is done by bandwidth probing, i.e. a cluster of packets is requested to be transmitted over the network to gauge if this will lead to increased delay and or loss. The implementation provides a function to do this measurement, enabling one to use this information to update the *pacing\_rate*.

The WebRTC implementation has defined additional API functions to monitor the states and statistics of the pacer.

##### 6.13.2.2.2 WebRTC and paced sender evaluation

To collect traffic characteristics, the following setup is deployed:

- In this case a call is started on a Microsoft edge browser and Wireshark is used to collect the traffic trace. The machine initiating the call is a Lenovo laptop with 16 GB of RAM running windows 11 operating system (referred to as machine A). The network is a Wifi connection on 2.4 GHz band setup by a Samsung Galaxy A54 5G connected to a 5G network using Wifi hotspot feature.

Figure 6.13.2.2.2-1 illustrates the traffic pattern measured from a WebRTC sender that joined a google hangout call on Machine A. This stream has a lower bit-rate as compared to the next experiment. The traffic is clearly separated in burst that corresponds to the frame transmission.

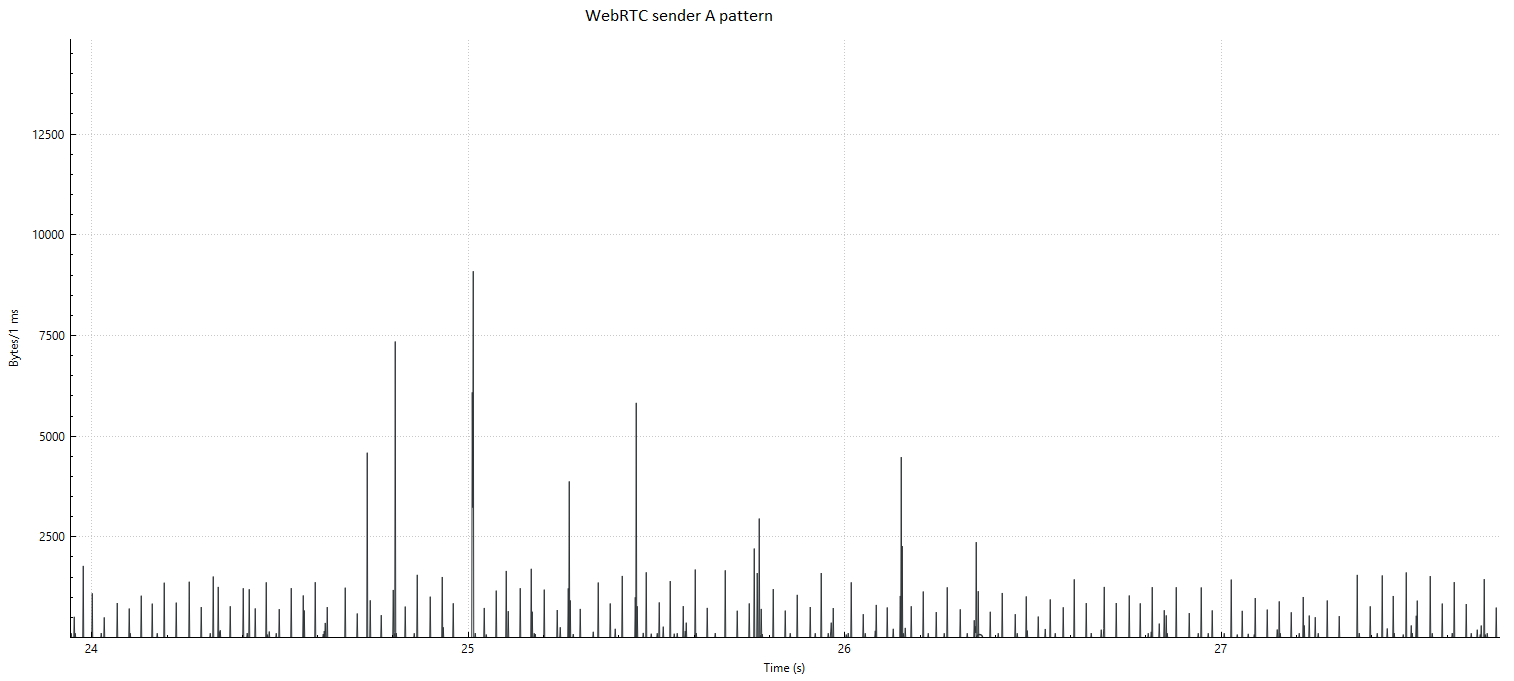


Figure 6.13.2.2.2-1: Sending pattern from WebRTC in a Google hangout call with video only

In a second experiment, we use a machine running Ubuntu 24.4 with 8 GB or RAM and Intel core i5 processor. The Ubuntu machine/Linux kernel enables throttling of the bandwidth and latency. The communication is via a Wifi and a 5G hotspot.

Linux traffic control was used using a token bucket filter (tbf) to slow down the data rate to 200 kbit and only enable 16 kbit bursts. The results are shown in Figure 6.13.2.2.2-2. We see the magnitude of the peaks is reduced as the hangout video quality drops after the traffic control filter is enabled. However the general traffic characteristics appear similar, there is no smoothening effect observed that may result from the paced sender. So, it seems in this case the general rate control of hangout for the video encoding reacted quick enough to avoid critical network operation.

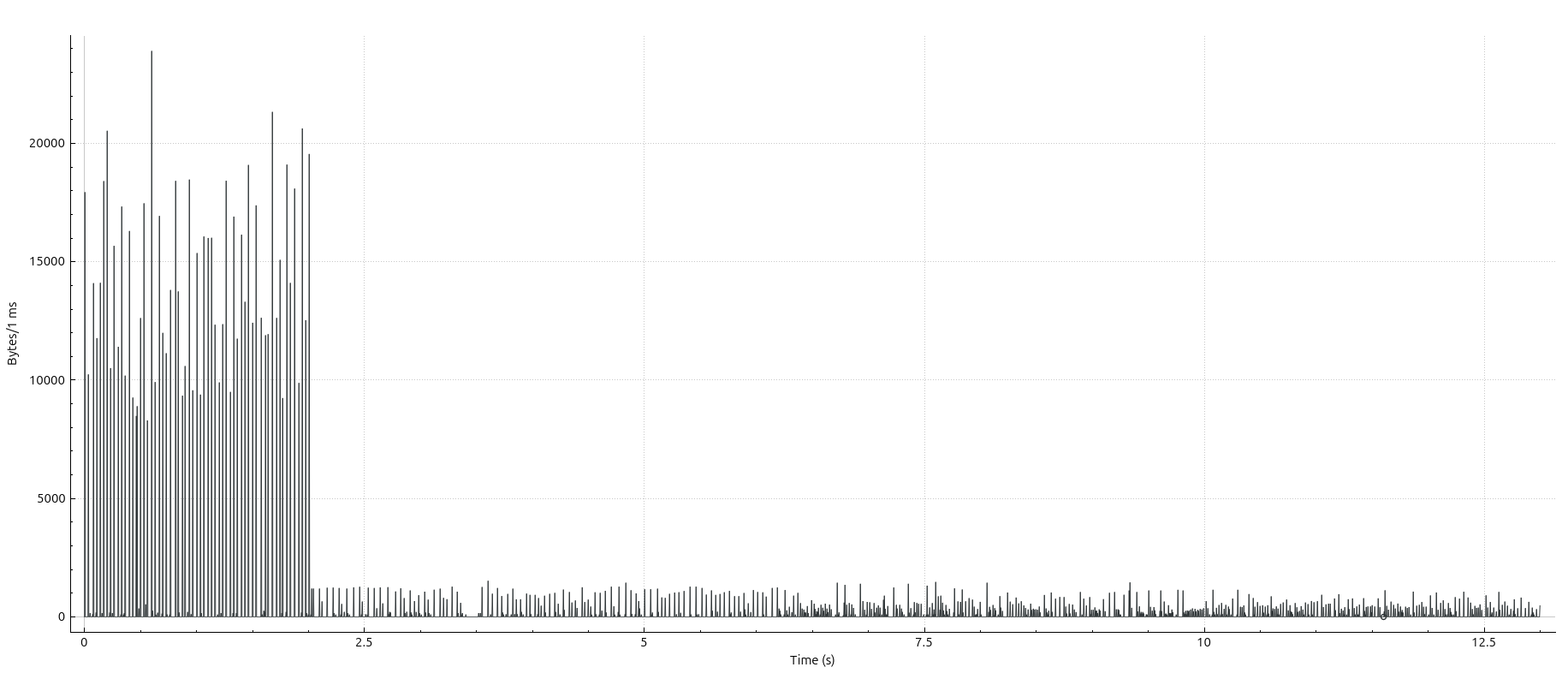


Figure 6.13.2.2.2-2: Resulting traffic characteristics at sender when the network bandwidth is throttled to 200 kbit audio + video

#### 6.13.2.3 GStreamer multimedia framework RTP implementation for P2P

6.13.2.3.1 GStreamer multimedia framework RTP implementation detailsGStreamer [39] is a library for constructing graphs of media-handling components. It supports a range of applications from simple playback to streaming up to complex mixing and editing workflows.

GStreamer works on all major operating systems such as Solaris, Unix etc.

It enables graph-based pipeline construction and has a broad coverage of different multimedia technologies for codecs, protocols, file format encapsulations etc.

The GStreamer 1.x API has been stable since 2012 and the library is quite likely to be found as a sub-component in many practical media systems implementations, especially open-source implementations.

In addition, the design of light weight data passing implies high performance and low latency, very applicable to real-time communications.

Due to the stability and long track record and wide deployment of this library, some sample pipelines utilizing the RTP features are developed in this study for collecting details on the traffic characteristics of this framework in simple RTP workflows.

The setup uses a camera followed by an H.264 encoder tuned for ultra-low latency and optimal speed settings, and the output is fed to an RTP pay-loader linked to a network sink. The receiver receives the RTP packets, decodes the streams and renders them to the screen enabling a 1-way conversational application.

##### 6.13.2.3.2 GStreamer RTP sender evaluation video

In this section the RTP video sender and receiver are evaluated.

To collect traffic characteristics, the following setup is used:

- Two Machines A and B (Lenovo and HP, both AMD Ryzen 5 processor, 16GB RAM are connected over Wifi network (2.4 GhZ) setup by a Galaxy A65 5G phone using hotspot function. Machine A runs the real time-frame capture, encoding,packetization and UDP transmission. Machine B receives the packets and depacketizes, decodes and renders the frame (receiver pipeline from the previous clause). Wireshark is used to collect the traffic data. In this setup a relatively high bit-rate of 16 MBps is used. GStreamer 1.24.5 is used for the experiment.

Figure 6.13.2.3.2-1 outlines the experimental setup.

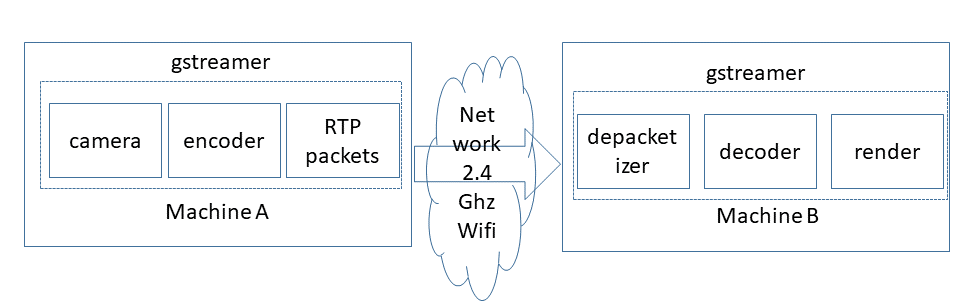


Figure 6.13.2.3.2-1: Experimental setup flow

Figure 6.13.2.3.2-2 illustrates the sending data traffic pattern in bytes per millisecond on a logarithmic scale. It is clear that the traffic is bursty and that the bursts occur on the timespan of 1-3 milliseconds.

Figure 6.13.2.3.2-3 illustrates the receiving data traffic pattern in bytes per millisecond on a logarithmic scale. The traffic bursty and that the burst occur in the timespan of 1-3 milliseconds, however already a little bit of dispersion can be seen due to transmission and delays experienced in the transmission.

Table 6.13.2.3.2-4 demonstrates a snapshot from the data collected from the sender A, one interesting observation is that the inter burst time is not constant. Between some burst there is approximately 20 milliseconds of idle period, while for other bursts the inter burst time is about 40-45 milliseconds.

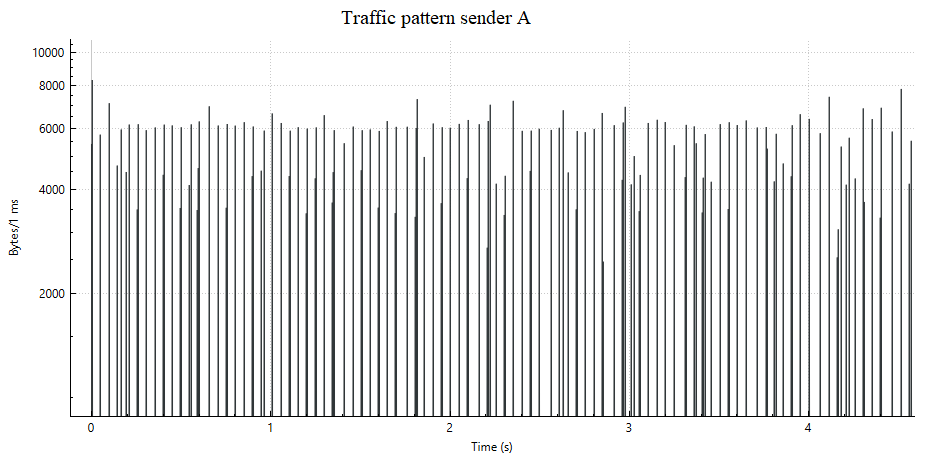


Figure 6.13.2.3.2-2: Traffic pattern in GStreamer RTP sender on machine A

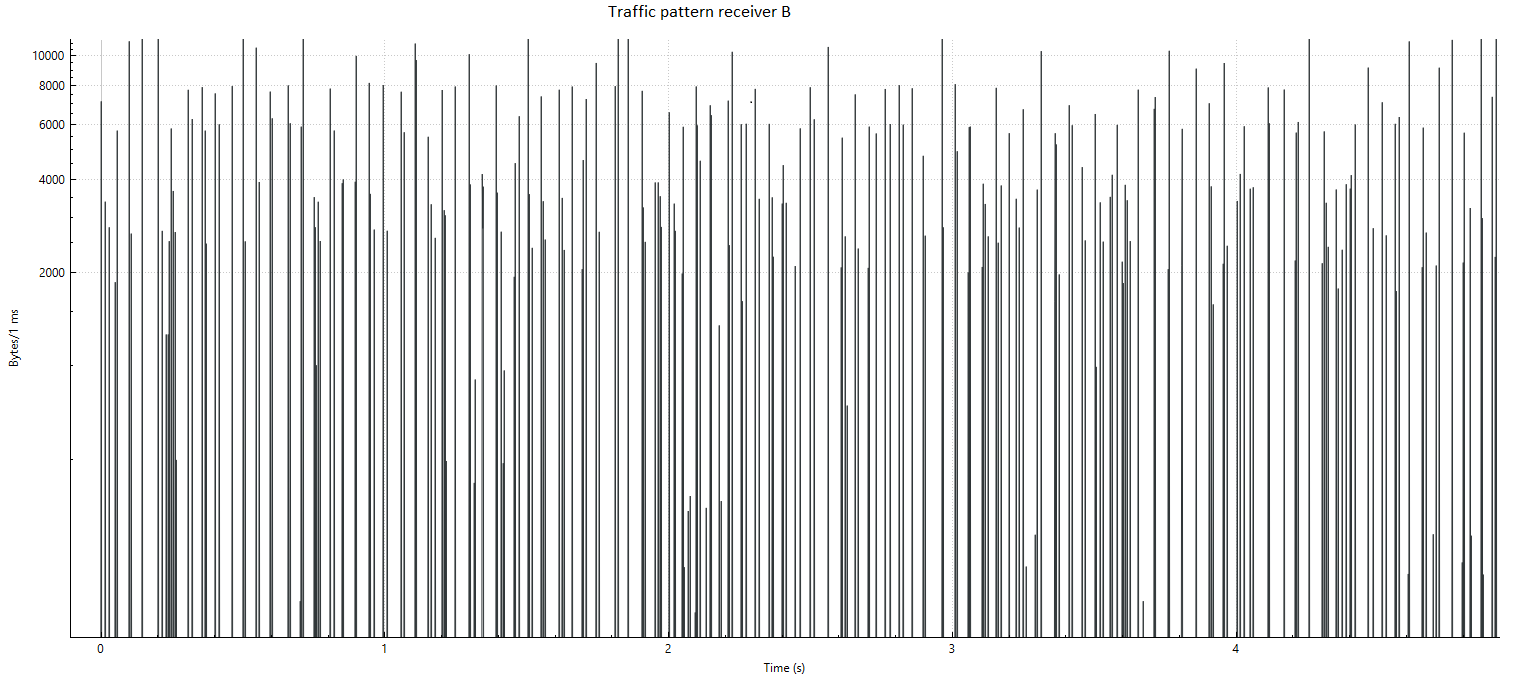


Figure 6.13.2.3.2-3: Traffic on machine B at the RTP receiver

Table 6.13.2.3.2-4: Snapshot of data from the sender, illustrating the inter-burst times

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| packet number | Time | Source IP | receiver IP |  | length |
| 108 | 0.192814 | 192.168.178.229 | 192.168.178.110 | UDP | 766 |
| 109 | 0.192830 | 192.168.178.229 | 192.168.178.110 | UDP | 767 |
| 110 | 0.192846 | 192.168.178.229 | 192.168.178.110 | UDP | 845 |
| 111 | 0.192859 | 192.168.178.229 | 192.168.178.110 | UDP | 520 |
| 112 | 0.192872 | 192.168.178.229 | 192.168.178.110 | UDP | 448 |
| 113 | 0.210483 | 192.168.178.229 | 192.168.178.110 | UDP | 56 |
| 114 | 0.210584 | 192.168.178.229 | 192.168.178.110 | UDP | 565 |
| 115 | 0.210627 | 192.168.178.229 | 192.168.178.110 | UDP | 985 |
| 116 | 0.210655 | 192.168.178.229 | 192.168.178.110 | UDP | 944 |
| 117 | 0.210687 | 192.168.178.229 | 192.168.178.110 | UDP | 1110 |
| 118 | 0.210716 | 192.168.178.229 | 192.168.178.110 | UDP | 1027 |
| 119 | 0.210742 | 192.168.178.229 | 192.168.178.110 | UDP | 758 |
| 120 | 0.210770 | 192.168.178.229 | 192.168.178.110 | UDP | 722 |
| 121 | 0.256040 | 192.168.178.229 | 192.168.178.110 | UDP | 56 |
| 122 | 0.256087 | 192.168.178.229 | 192.168.178.110 | UDP | 317 |
| 123 | 0.256104 | 192.168.178.229 | 192.168.178.110 | UDP | 416 |
| 124 | 0.256125 | 192.168.178.229 | 192.168.178.110 | UDP | 584 |
| 125 | 0.256143 | 192.168.178.229 | 192.168.178.110 | UDP | 812 |

##### 6.13.2.3.3 Gstreamer RTSP sender/receiver evaluation PCM audio + video

In this section the RTP video + audio sender and receiver are evaluated.

The video is as in previous section, while the audio is transmitted as PCM audio.

To collect traffic characteristics, the following setup is used:

- Two Machines A and B (Lenovo and Asus, first with AMD Ryzen 5 processor and second with Intel i5, 16 GB RAM/8 GB RAM are connected over Wifi network (2.4 GHz) setup by a Galaxy A65 5G phone using hotspot function. Machine A runs the real time-frame capture, encoding packetization and UDP transmission (sender pipeline from the previous clause). Machine B receives the packets and de-packetizes, decodes and renders the frame (receiver pipeline from the previous clause). Wireshark is used to collect the traffic data.

The results are shown in the figures below, the dynamic traffic characteristics bear similarity with those in the previous section, but the audio packets are much more frequent peaks with a smaller size.

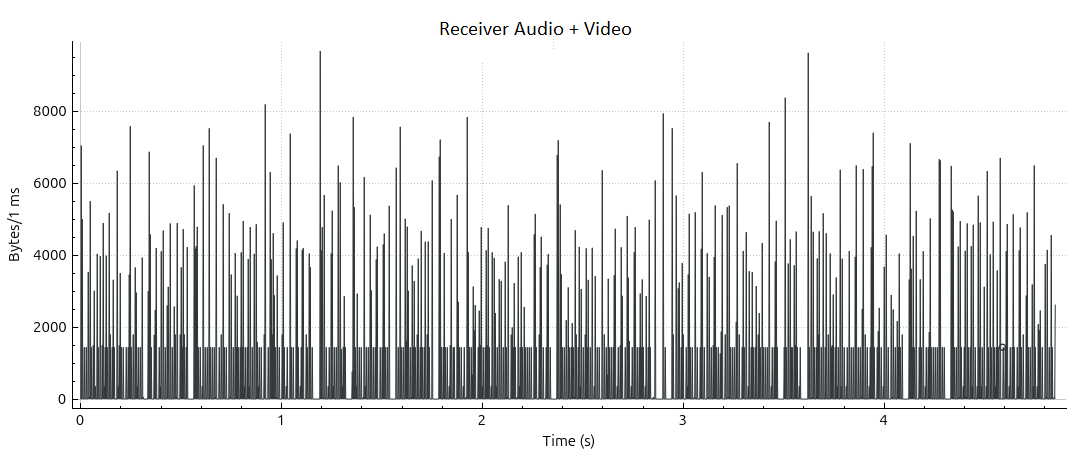


Figure 6.13.2.3.3-1: Receiver audio + video

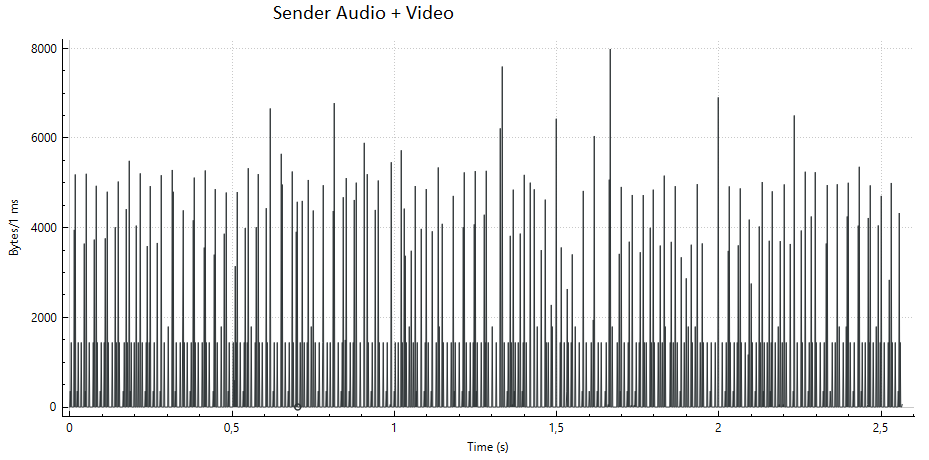


Figure 6.13.2.3.3-2: Sender audio + video

#### 6.13.2.4 Server-side senders

##### 6.13.2.4.1 Server sender implementation details

In this clause we consider server implementations of RTP senders using RTSP and/or RTP.

Both a popular commercial server and a popular open-source server implementation are evaluated.

The commercial server used for evaluation is Wowza Streaming Engine™ [40] (before Wowza Media Server™), one of the popular streaming servers that can support a wide range of transmission protocols. Wowza Streaming Engine™ is a Java application that runs as a service. The server can be used for a variety of live and on-demand streaming applications and it supports RTP/RTSP and low latency delivery.

The open-source server used in this evaluation is the popular Media MTX (before called simple RTSP server) [41], Media MTX is a real time media server/proxy that can support different protocols, including RTSP and RTP, but also WebRTC and RTMP [42]. The GitHub statistics on [41] show this as one of the most popular open source RTP/RTSP server implementations. It is possible, similar as in Wowza to ingest camera and other streams into the server for redistribution.

##### 6.13.2.4.2 Server side implementation evaluation

Wowza Streaming Engine™ version 4.8.27 was used deployed on a Lenovo laptop with 16 GB or RAM with AMD Ryzen 5. Another laptop with Intel i7-6600 2.6 GHz CPU and 16 GB of RAM is used to receive the stream from the server. A camera stream is ingested over RTP to the Wowza Streaming Engine™ using a GStreamer pipeline similar as the P2P sender in the previous section. In Wowza Streaming Engine™ a predefined session description file is used to ingest the stream. The stream is distributed using RTP/RTSP. The machines are connected over a 2.4 GHz wifi network setup by a Galaxy A54 5G phone using the hotspot function. The receiver laptop uses VLC media player 3.0.8 for receiving the stream using RTSP and RTP.

Media MTX version 1.8.24 was used deployed on a Lenovo laptop with 16GB or RAM with AMD Ryzen 5. Another laptop Microsoft surface with Intel i7-6600 2.6 GHz CPU and 16GB of RAM was used to receive the stream. A camera stream is ingested over RTP to the wowza streaming engine using Gstreamer, instead of RTP + SDP in this case WebRTC + WHIP was used for the ingest, which automizes the session description exchange needed for the ingest. The stream is distributed by Media MTX engine using RTP/RTSP. The machines are connected over a 2.4 GHz Wifi network setup by a Galaxy A54 5G phone using the hotspot function. The receiver laptop uses VLC media player 3.0.8 for receiving the stream over RTSP.

The flow of the experiment can be shown in Figure 6.13.2.4.2-1, GStreamer is used for the camera ingest to MTX Media or Wowza Streaming Engine™, then the data is transmitted to the client by the server, and Video Lan Client (VLC) is used to receive the stream and play it back. In this scenario RTSP + RTP is used.

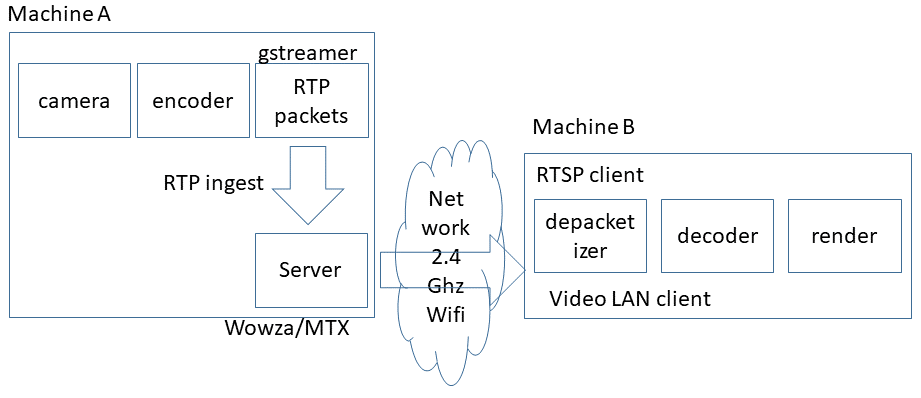


Figure 6.13.2.4.2-1: Setup of the experiment using server

The results for the different configurations are plotted in Figures 6.13.2.4.2-2 - 6.13.2.4.2-6 collecting traffic measurements. Only short intervals of time are measured, and the measurements from sender and receiver do not correspond timewise. The main point of investigation are the dynamic traffic characteristics relating to the burst size and periodicity, in each of the setups the burstiness corresponding to frame transmissions can be observed. It seems the MTX media the times between the bursts vary a bit more compared to the Wowza setup.

From the experiments it seems the lower framerate mostly results from the camera capture, tests with an artificial test source generating content at higher framerateresults in the higher burst frequency (see Figure 6.13.2.4.2-4).

In Figure 6.13.2.4.2-6 an artificial video source is used to ingest to MTX with a higher framerate and quality. The results show that in this case the burst frequency of traffic transmitted by MTX media is also increased.

In addition, in 6.13.2.4.2-7 also shows when both audio (PCM) and video (H.264) are transmitted from MTX media server, in this case we see the audio bursts are more frequent and smaller in size compared to the video burst.

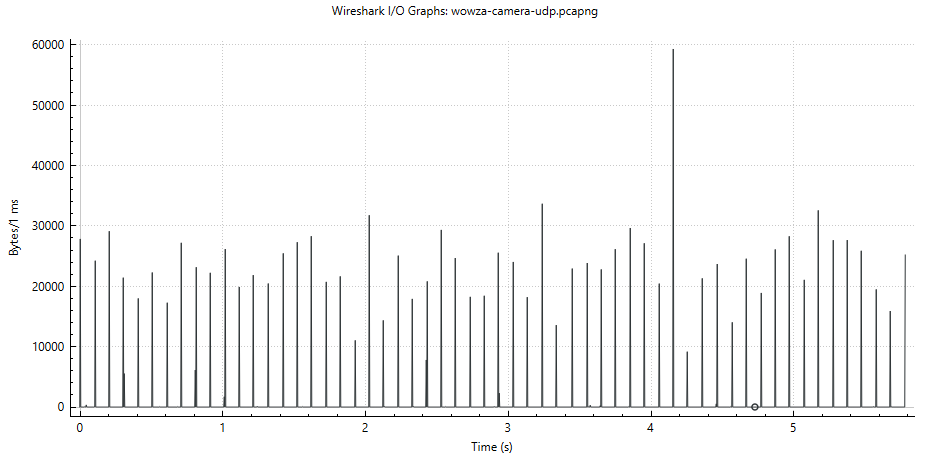


Figure 6.13.2.4.2-2: Wowza server side traffic sending to a VLC receiver using RTSP/RTP

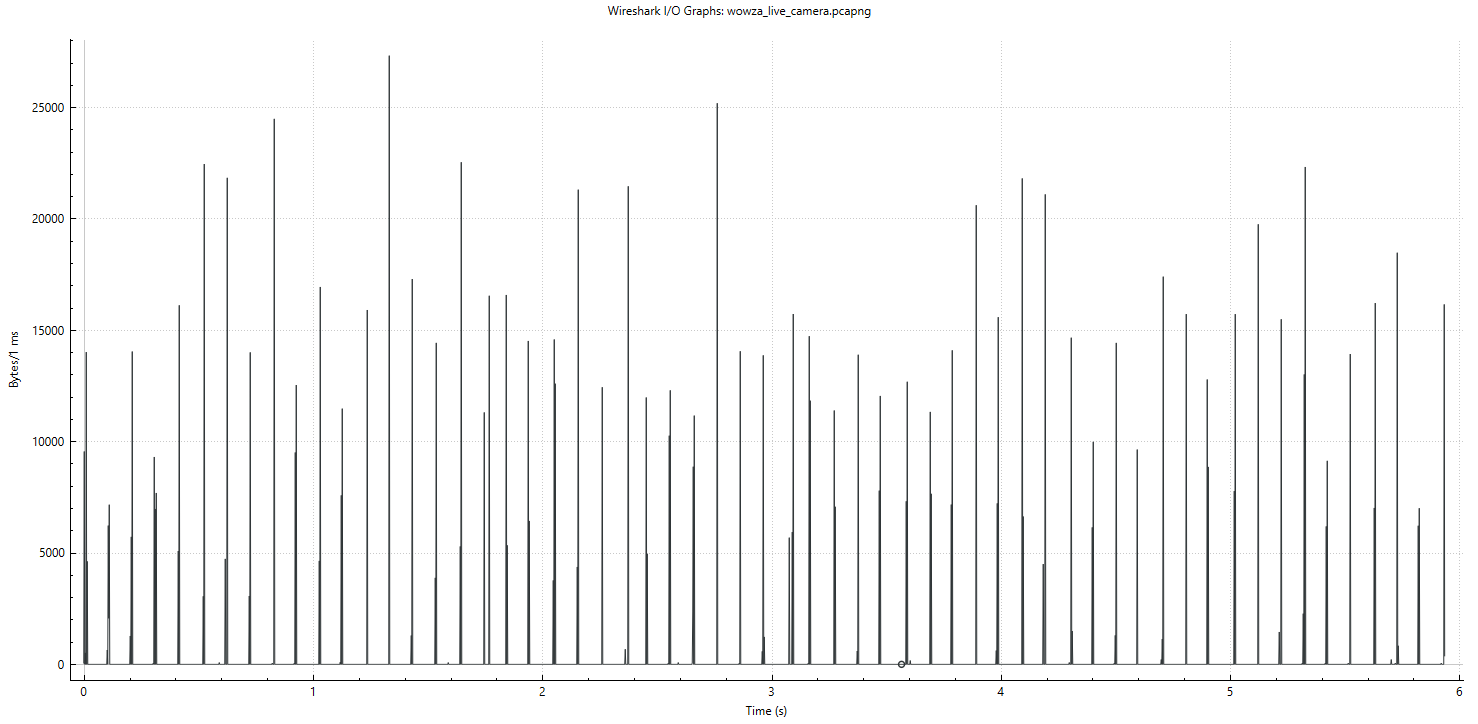


Figure 6.13.2.4.2-3: VLC receiver side traffic from received RTP packets from Wowza Streaming Engine™

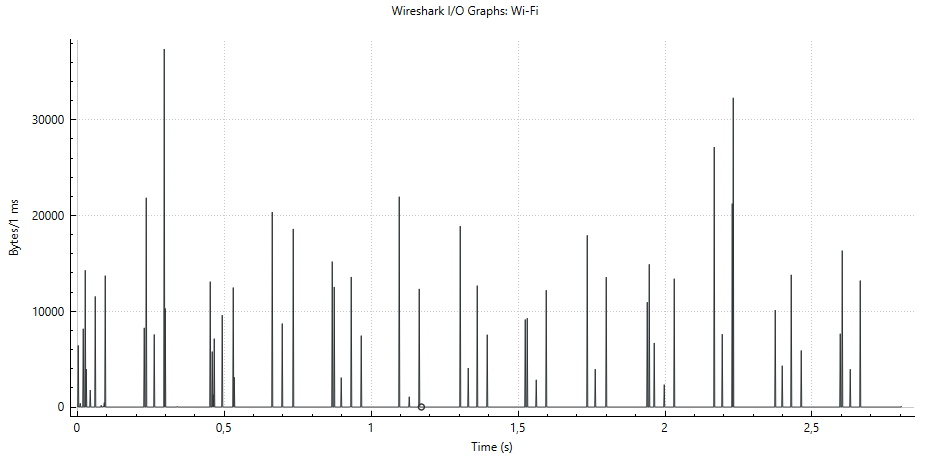


Figure 6.13.2.4.2-4: Camera RTP Traffic monitored from sender from MTX Media

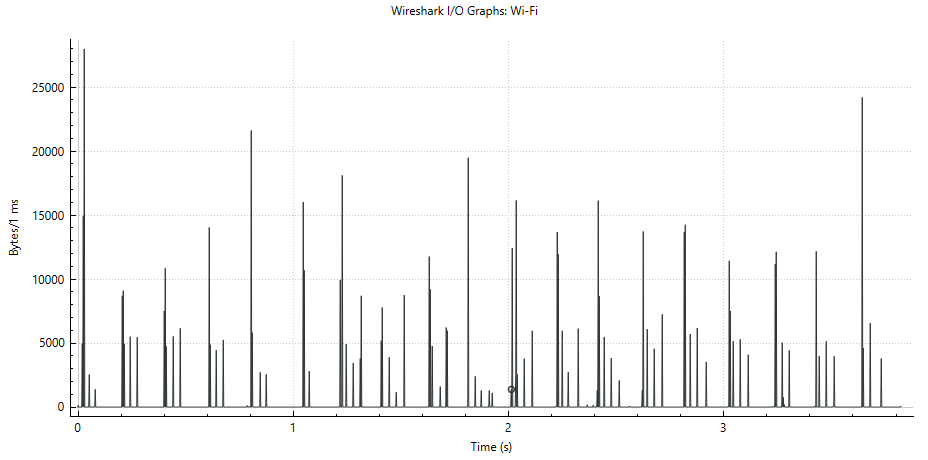


Figure 6.13.2.4.2-5: Camera RTP traffic monitored at VLC receiver from MTX Media

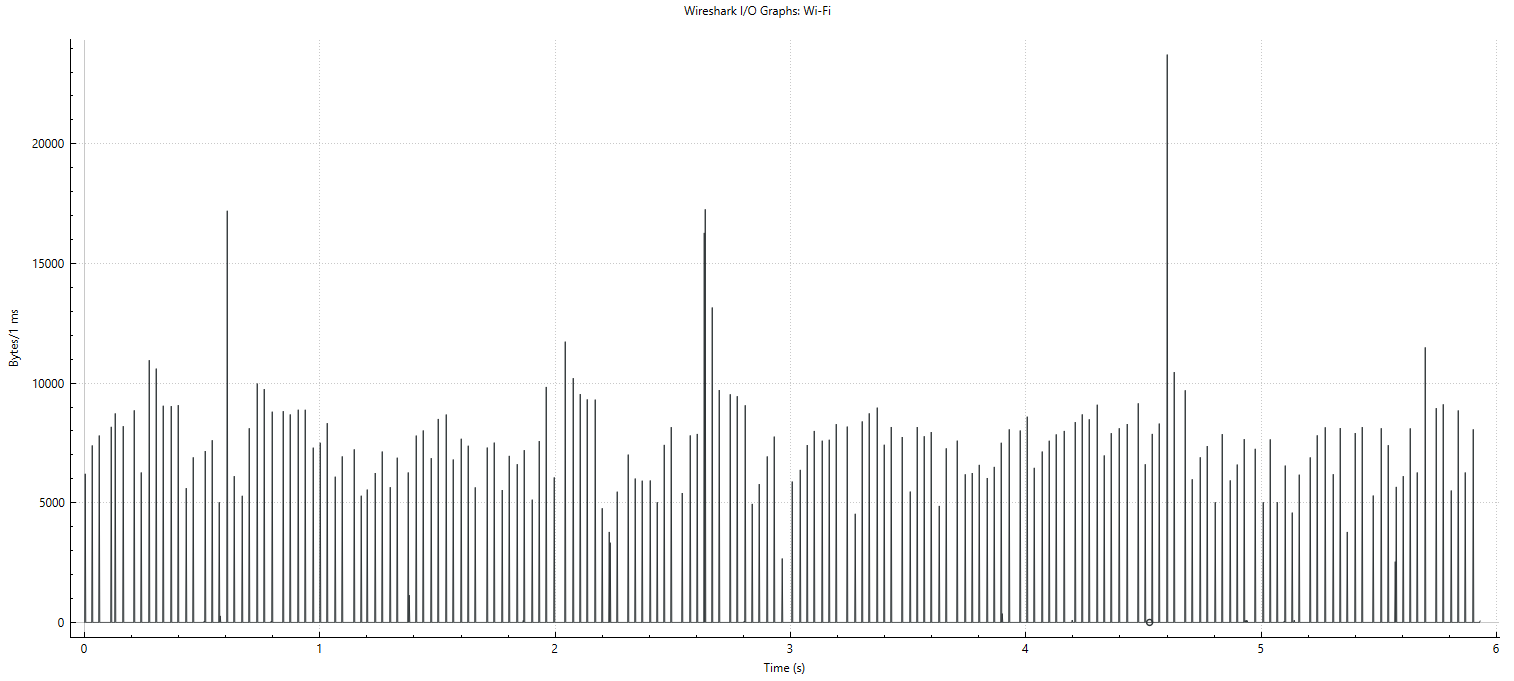


Figure 6.13.2.4.2-6: Synthetic test source RTP Traffic monitored from sender from MTX Media server

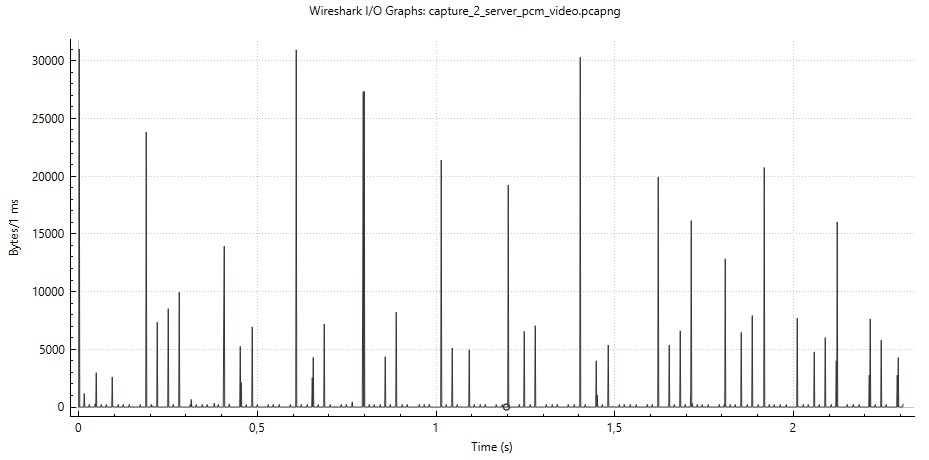


Figure 6.13.2.4.2-7: Traffic pattern from MTX sender server audio + video

#### 6.13.2.5 Aggregate Statistics

This clause presenta some aggregate statistics for dynamic traffic characteristics are retrieved from the packet traces collected in this solution.

Table 6.13.2.5-1 illustrates the identified metrics.

Deterministic metrics, averages, standard deviations and percentages are considered.

The metrics are coded with a short abbreviation that is used when reporting in subsequent tables.

In Table 6.13.2.5-2 the scenarios described in the previous clauses are summarized and coded with a short abbreviation that is used then reporting the aggregate statistics in subsequent tables.

Table 6.13.2.5-3 illustrates the results of the metrics derived from the packet traces.

For brevity we can leave out the deterministic metrics bt, tbs, #bp, #lp, #bb and #lb that depend heavily on the length of the trace. Instead the focus is on the ratio between lone packets/bytes and burst packets/bytes and other metrics that are independent of the length of the trace.

Some observations of the statistics in Table 6.13.2.5-3:

- A large percentage of the traffic and packets are part of a burst, highlighting the relevance of burst handling

- Burst durations in the traces are shorter than a millisecond

- PCM audio is tested, transmitted in lone packets, this increases the share of lone bytes (compared to more efficiently compressed audio)

Table 6.13.2.5-1 Metrics and statistics considered for dynamic traffic characteristics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Metric** | **meaning** | **Additional notes** | **Statistics** |
| Bt | Burst Time (Bt) | The time of the burst (corresponding to the first packet), measured in seconds | Bursts are measured as instantaneous transmission of 2 or more packets | Deterministic |
| Ibt | Inter Burst Time (Ibt) | The average duration between 2 consecutive bursts (burst), measured in seconds | This does not consider single ‘lone’ packets transmitted between the bursts | Average and Standard Deviation |
| Bd | Burst Duration (Bd) | The duration between start and end of a burst, measured in seconds | Measures the duration of the burst (in a maximum timeslot of 2 milliseconds ) | Average and Standard Deviation |
| Bs | Burst Size (Bs) | The Size of the Data Burst measured in bytes | Measures the size of the entire burst (in a maximum timeslot of 2 milliseconds) | Average and Standard Deviation |
| Ppb | Packets per burst (Ppb) | Number of packets in each burst | Only bursts of 2 packets or more are considered | Average and Standard deviation |
| #bp | Total Burst packets (#bp) | Number of packets in a burst | This is the number of packets in the trace that is part of a burst | Deterministic |
| #lp | Total Lone packets (#lp) | Number of packets not in a burst | This is the number of packets in the trace that is part not part of a burst | Deterministic |
| #bb | Total Burst Bytes (#bb) | Number of bytes that are part of a burst | This is the number of bytes that is part of a burst | Deterministic |
| #lb | Total Lone Bytes (#lb) | Number of bytes that are not part of a burst | This is the number of bytes in lone packets | Deterministic |
| %lp | Percentage of lone packets (%lp) | Ratio of lone packets to all packets |  | Percentage |
| %bp | Percentage of burst packets (%bp) | Ratio of burst packets to all packets |  | Percentage |
| %lb | Percentage of lone bytes (%lb) | Ratio of lone packet bytes to all bytes |  | Percentage |
| %bb | Percentage of burst bytes (%bb) | Ratio of burst packet bytes to all bytes |  | Percentage |

Table 6.13.2.5-2 Scenario descriptions and coding in solution #13

|  |  |
| --- | --- |
| Scenario code |  |
| w\_h | Windows hangout without network throttling duplex case (Figure 6.13.2.2.2-1) [1] |
| u\_h | Ubuntu Hangout with network throttling simplex case (Figure 6.13.2.2.2-2)[1] |
| g\_v\_a | Gstreamer video only measured on machine A (Figure 6.13.2.3.2-2) [1] |
| g\_v\_b | Gstreamer video only measured on machine B (Figure 6.13.2.3.2-3) [1] |
| g\_va\_a | Gstreamer pcm audio + video at receiver (Figure 6.13.2.3.3-1) [1] |
| g\_va\_b | Gstreamer pcm audio + video at sender (Figure 6.13.2.3.3-2)[1] |
| w\_s | Wowza at server measured at server (Figure 6.13.2.4.2-2) [1] |
| w\_r | Wowza stream measured at player receiver (Figure 6.13.2.4.2-3) [1] |
| m\_rv | MTX server measured vlc player receiver (Figure 6.13.2.4.2-5) [1] |
| m\_sav | MTX server video + PCM (Figure 6.13.2.4.2-7) [1] monitored at server |
| m\_rav | MTX server video + PCM (Figure 6.13.2.4.2-7) [1] monitored at vlc client |

Table 6.13.2.5-3 Statistical results of experiments in different scenarios in scenario #13

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Scenario  (direction) | ibt  (mean) | Ibt  (stdev) | Bd  (mean) | Bs  (mean) | Ppb  (mean) | %lp  (%) | %bp  (%) | %lb  (%) | %bb  (%) |
| w\_h (a) | 0.0352 | 0.01437 | 0.0002 | 3463 | 3.55 | 23.2 | 76.7 | 6.6 | 93.4 |
| w\_h (b) | 0.01592 | 0.019 | 0.00016 | 3756 | 3.3 | 19.2 | 80.8 | 17.4 | 82.6 |
| u\_h (a) | 0.03132 | 0.00873 | 0.00017 | 13895 | 12,3 | 40.2 | 59.8 | 22.6 | 77.4 |
| u\_h (b) | 0.42321 | 0.37918 | 0.00007 | 213.92 | 2 | 72.2 | 27.8 | 73.8 | 26.2 |
| g\_v\_a | 0.01642 | 0.01175 | 0.00052 | 2864 | 7.73 | 2.4 | 97.6 | 0.7 | 99.3 |
| g\_v\_b | 0.01796 | 0.00967 | 0.00029 | 3109 | 8.6 | 0.2 | 99.8 | 0.2 | 99.8 |
| g\_va\_a | 0.01252 | 0.0114 | 0.00071 | 4144 | 6.6 | 14.2 | 85.8 | 20.1 | 79.9 |
| g\_va\_b | 0.01414 | 0.00955 | 0.00033 | 4472 | 7.5 | 14.8 | 85.2 | 24.7 | 75.3 |
| w\_s (a) | 0.08732 | 0.03483 | 0.00079 | 24822 | 42.3 | 0.2 | 99.8 | 0.0 | 100.0 |
| w\_r (a) | 0.03778 | 0.04627 | 0.00054 | 10671 | 18.24 | 0.7 | 99.3 | 0.1 | 99.9 |
| m\_rv | 0.02749 | 0.0404 | 0.00084 | 6630 | 6.38 | 2.5 | 97.5 | 0.5 | 99.5 |
| m\_sav (a) | 0.04777 | 0.03838 | 0.00056 | 11906 | 12.18 | 12.6 | 87.4 | 3.3 | 96.7 |
| m\_rav (a) | 0.02573 | 0.03412 | 0.00086 | 6692 | 6.8 | 12.2 | 87.8 | 3.2 | 96.5 |

### 6.13.3 Discussion and conclusion

**Observation 1:**

The data from the GStreamer RTP and from WebRTC browser implementation show bursts of data being transmitted in short time intervals. There is no evidence that the encoder/packager gradually produces and transmits packets, as in the experiments only short bursts were observed. It could happen on a per frame(s) basis in a short time interval. It may also be that some frames are combined in a single burst, but this cannot be observed from the current results.

Similar observations were made when an intermediate server was used for redistributing the streams.

It should therefore be possible for these applications to calculate the burst size a priori before sending it out. The added latency should be limited to around a few milliseconds even in the worst case.

In this scenarios large percentage of the total traffic (packets/bytes) is part of a burst.

In this study only burst of size more than 1 packet were considered.

The results show that in cases a large percentage of the bytes/packets (generally over 90%) are transmitted in bursts.

This highlights that improvement of transmission of bursty traffic can be beneficial as this is such a large percentage of traffic.

The dynamic traffic characteristics could be the

NOTE 1: The latency is related to sending the packets on the network, not to the encoder or RTP packager generating the packets. Detailed study of the cause of the delay is FFS. The preliminary conclusion is that a few milliseconds of delay may be introduced is only a worst-case estimate.

**Observation 2:**

The inter-burst time interval seems regular, but not constant. Therefore, signalling of the time to next burst, if known by the application, may be suitable for signalling as a dynamic traffic characteristic.

Aggregate metrics for inter burst times have been derived (average/standard deviation), but these do not relate to the potential to predict or measure these in an application.

NOTE 2: This requires more study of different patterns and situations.

**Observation 3:**

It can be derived that when the network is not overloaded the WebRTC and GStreamer implementations do not differ too much and the influence of the paced sender module is limited.

In addition, when the network bandwidth changes all of a sudden, the rate control may adapt so well that the effect of the paced sender is not significant.

**Observation 4 (extra on P2P versus Server based):**

When comparing the end-end latency of the peer-to-peer setups to the server based setups, for the peer to peer setups latencies < 1 seconds are achieved while in the server routing setup latencies of around 3 seconds are observed.

**Conclusion:**

Short periodic traffic bursts in short intervals occur in typical real time conversational applications using real-time video + audio.

Given the observed traffic behaviour and the observed application behaviour it seems achievable to include information about a burst size before it is being sent out as the durations of the burst are in the order of less than 1-2 milliseconds (mostly less than 1 millisecond), but it may require some changes in the sender implementation to achieve this.

A large percentage of the transmitted bytes/packets are part of a data burst.

### 6.13.4 Conclusion

Given the observed behaviour of common RTP Sender implementations, the conclusion and recommendation can be made as following:

- extend the signaling of dynamic traffic characteristics to signal data burst size and time to next burst in the RTP header extension.

- complementary signaling/additions to the above RTP header extension.

- Transmission of bursty traffic is important as in RTP senders a large percentage of the bytes/packets (> 90%) is transmitted in bursts.

|  |
| --- |
| END of CHANGES |