**Agenda item:** 10.6

**Source:** Qualcomm Inc.

**Title: [5G\_RTP\_Ph2] Impact of WebRTC paced sending on burst duration**

**Document for** Discussion andAgreement

# Introduction

There has been discussion about paced sending in WebRTC. It has been reported in clause 16 of [1] that the data burst duration is less than 1ms for some scenarios.

We discuss whether the bursty traffic pattern reported above can be generalized.

# Analysis of WebRTC source code

Extremely bursty traffic patterns are intended to be avoided as discussed below in the WebRTC document [2], because such bursty traffic may have detrimental effects to the network, causing buffer bloat or packet losses, as stated below:

“

Consider a video stream at 5Mbps and 30fps. This would in an ideal world result in each frame being ~21kB large and packetized into 18 RTP packets. While the average bitrate over say a one second sliding window would be a correct 5Mbps, on a shorter time scale it can be seen as a burst of 167Mbps every 33ms, each followed by a 32ms silent period. Further, it is quite common that video encoders overshoot the target frame size in case of sudden movement especially dealing with screensharing. Frames being 10x or even 100x larger than the ideal size is an all too real scenario. These packet bursts can cause several issues, such as congesting networks and causing buffer bloat or even packet loss. …

The paced sender solves this by having a buffer in which media is queued, and then using a *leaky bucket* algorithm to pace them onto the network.

”

**Conclusion 1:** Traffic that is very bursty can cause issues to the network such as buffer bloat or even packet losses.

The leaky bucket algorithm limits the burstiness, and the implementation in WebRTC allows a sender to send at higher data rate in a short period of time while maintaining a lower average data rate in the long term. In WebRTC, the lower long-term average data rate is provided by the congestion control algorithm and is also used in determining the higher data rate, which is also known as the pacing rate. For Google Congestion Control, the higher data rate by default is set to pacing\_factor\_ = 2.5 times of the maximum data rate allowed by congestion control [3]. This is shown in the code snippet below:

// Pacing-rate relative to our target send rate.

// Multiplicative factor that is applied to the target bitrate to calculate

// the number of bytes that can be transmitted per interval.

// Increasing this factor will result in lower delays in cases of bitrate

// overshoots from the encoder.

constexpr float kDefaultPaceMultiplier = 2.5f;

 …

 pacing\_factor\_(config.stream\_based\_config.pacing\_factor.value\_or(

 kDefaultPaceMultiplier)),

 …

PacerConfig GoogCcNetworkController::GetPacingRates(Timestamp at\_time) const {

 // Pacing rate is based on target rate before congestion window pushback,

 // because we don't want to build queues in the pacer when pushback occurs.

 DataRate pacing\_rate = DataRate::Zero();

 if (pace\_at\_max\_of\_bwe\_and\_lower\_link\_capacity\_ && estimate\_ &&

 !bandwidth\_estimation\_->PaceAtLossBasedEstimate()) {

 pacing\_rate =

 std::max({min\_total\_allocated\_bitrate\_, estimate\_->link\_capacity\_lower,

 last\_loss\_based\_target\_rate\_}) \*

 pacing\_factor\_;

 } else {

 pacing\_rate =

 std::max(min\_total\_allocated\_bitrate\_, last\_loss\_based\_target\_rate\_) \*

 pacing\_factor\_;

 }

**Observation 1:** In WebRTC, the data rate during the transmission of a burst is set to 2.5 times of the maximum data rate allowed by congestion control by default.

When the packets are sent in a burst, the packets are paced out one by one at the pacing rate (to be precise, an adjusted pacing rate called adjusted\_media\_rate\_). The pacing algorithm is implemented in pacing\_controller.cc [4], shown in the code snippet next. In Line 37, the target time of sending the next packet is computed by calling NextSendTime(), which is defined in Line 40 and onwards and returns next\_send\_time. The variable media\_debt\_ is the size of packets in the queue yet to be transmitted.

void PacingController::ProcessPackets() {

…

 target\_send\_time = NextSendTime();

}

Timestamp PacingController::NextSendTime() const {

…

 if (adjusted\_media\_rate\_ > DataRate::Zero() && !packet\_queue\_.Empty()) {

 // If packets are allowed to be sent in a burst, the

 // debt is allowed to grow up to one packet more than what can be sent

 // during 'send\_burst\_period\_'.

 TimeDelta drain\_time = media\_debt\_ / adjusted\_media\_rate\_;

 // Ensure that a burst of sent packet is not larger than kMaxBurstSize in

 // order to not risk overfilling socket buffers at high bitrate.

 TimeDelta send\_burst\_interval =

 std::min(send\_burst\_interval\_, kMaxBurstSize / adjusted\_media\_rate\_);

 next\_send\_time =

 last\_process\_time\_ +

 ((send\_burst\_interval > drain\_time) ? TimeDelta::Zero() : drain\_time);

 }

…

}

Based on the above source code in [3] and [4], we have the following observation.

**Observation 2:** In WebRTC, the burst duration depends on the maximum sending rate allowed by congestion control.

With a default scaling factor of 2.5 on the pacing rate, if the long-term video bit rate is equal to the maximum sending rate allowed by congestion control, the burst duration will be 1/(2.5) = 40% of the inter-frame period on average. This translates into an average burst duration of 16.7×40%=6.68ms for 60fps video. If the long-term video bit rate is lower than the sending rate, the burst duration will be shorter than 6.68ms.

The above calculation and Conclusion 1 lead to the following conclusion.

**Conclusion 2**: In WebRTC, the burst duration is not necessarily limited to a particular value such as 1ms.

# Summary

Based on the analysis of the WebRTC source code, we arrive at the following observations and conclusions.

**Conclusion 1:** Traffic that is too bursty can cause issues to the network such as buffer bloat or even packet losses.

**Observation 1:** In WebRTC, the data rate during the transmission of a burst is set to 2.5 times of the maximum data rate allowed by congestion control by default.

**Observation 2:** In WebRTC, the burst duration depends on the maximum sending rate allowed by congestion control.

**Conclusion 2**: In WebRTC, the burst duration is not necessarily limited to a particular value such as 1ms.

# References

[1] TR26.822, Study on 5G Real-time Transport Protocol Configurations, V1.2.0, Rel-19, Nov 2024.

[2] Paced Sending description, [https://webrtc.googlesource.com/src/+/refs/heads/main/modules/pacing/g3doc/index.md](https://webrtc.googlesource.com/src/%2B/refs/heads/main/modules/pacing/g3doc/index.md), retrieved on February 8, 2025.

[3] goog\_cc\_network\_control.cc [https://source.chromium.org/chromium/chromium/src/+/main:third\_party/webrtc/modules/congestion\_controller/goog\_cc/goog\_cc\_network\_control.cc](https://source.chromium.org/chromium/chromium/src/%2B/main%3Athird_party/webrtc/modules/congestion_controller/goog_cc/goog_cc_network_control.cc), retrieved on February 8, 2025.

[4] Paced Sending Code: [https://webrtc.googlesource.com/src/+/refs/heads/main/modules/pacing](https://webrtc.googlesource.com/src/%2B/refs/heads/main/modules/pacing), retrieved on February 8, 2025.