




Exploration and Evaluation of Self-Clocked Rate Adaptation for Multimedia (SCReAM) Congestion Control Algorithm in 5G Networks

Ahmed Samir Jagmagji ^{*,†,1}, Haider Dhia Zubaydi ^{*,2}, and Sándor Molnár ^{*,3}

^{*} Department of Telecommunications and Media Informatics (TMIT)
Faculty of Electrical Engineering and Informatics (VIK)
Budapest University of Technology and Economics
Budapest, Hungary

[†] Department of Computer Engineering, College of Engineering
University of Mosul, Mosul, Iraq

ahmedjagmagji1983@edu.bme.hu ¹ , haider.zubaydi@tmit.bme.hu ² , molnar@tmit.bme.hu ³ 

Abstract—The new emerging mobile technologies have led to increased complexity in the networking paradigm. These technologies are incorporated into each device to support today's trends. Although this process offers many advantages, it brought several issues when new use-cases and services are introduced such as requiring regular updating of infrastructure, high network throughput, low latency, and minimum packet loss to provide proper performance. One of the main interesting applications that can benefit from such features is the conversational video for WebRTC which is becoming more popular since the developing community is growing and the industry is expressing its interest by striving towards designing new schemes for it. The main concept that controls the conversational video for WebRTC is the congestion control algorithm. Many approaches have been designed to address certain issues to result in optimum performance. In this study, we aim to explore the possibility of achieving optimal streaming in a 5G environment. We are especially interested in choosing the appropriate congestion control algorithm to achieve this purpose. As a candidate, we are focusing on the Self-Clocked Rate Adaptation for Multimedia (SCReAM) congestion control algorithm. Also, we aim to provide guidelines for achieving the video streaming performance of SCReAM in 5G networks.

Keywords—5G, Congestion Control, Conversational Video, mmWave, Network, TCP, WebRTC

I. INTRODUCTION

Current video streaming applications have gained the interest of various industries due to their ability to create a comprehensive reality and get people close to each other. By utilizing IP protocols, such applications allow for optimum end-to-end (E2E) experience by offering deliverance assurance and the best reachable Quality of Service (QoS) [1]. In computing networks, wired and wireless communications are being designed to offer compatibility with different technologies and the ability to process services at high speeds. Constraints on available resources (such as bandwidth) cause packet loss and re-transmission when the network is overloaded, which refers to “congestion” [2].

Network congestion occurs when the network is unable to handle heavy traffic which leads to reduced response time or in some cases, collapses the network. Hence, network congestion causes a greater impact and needs further consideration. Furthermore, all media traffic especially the audiovisual is increasing due to networking applications that are built on top of the transport layer which utilizes the network to run Video on Demand (VoD), Voice Over IP (VoIP), and video conferencing [3]. Although IETF has spent enormous efforts to standardize WebRTC for real-time applications such as Real-Time Transport Protocol (RTP), the industry is still using its own proprietary algorithm and protocols for their products because it has been reluctant to use standard protocol approaches [4].

Based on the above-mentioned details and due to the constant changes in current networking paradigms, the volume of the generated traffic is much higher than before. It is important to consider network congestion as one of the main issues nowadays. Congestion control algorithms are now a critical part of any network design and management to eliminate any effect on users' experience. Many congestion control algorithms are introduced to address the previously mentioned issues. SCReAM [5], [6] is an efficient congestion control algorithm that has been proposed in 2014 for conversational video for LTE networks. Later, it has been standardized in 2017 (RFC8298). Currently, there is not a clear guide on how to choose the best parameters of SCReAM to result in optimum performance. Also, it is important to realize the expected performance of SCReAM in 5G environments.

In this study, we aim to explore the possibility of achieving optimal streaming in a 5G environment. We are especially interested in choosing the appropriate congestion control algorithm to achieve this purpose. As a candidate, we are focusing on the Self-Clocked Rate Adaptation for Multimedia (SCReAM) congestion control algorithm. Also, we aim to provide guidelines for achieving the video streaming performance

of SCReAM in 5G networks.

Many parameters were investigated in order to implement SCReAM in the appropriate 5G environment. First, we implemented this work in a Windows-based test application by setting the initial parameters. Then, we tested SCReAM with default settings to ensure that it has been implemented properly. Later, we modified the algorithm to be used in a 5G environment through the retrieved 5G datasets. This supports our further research phases to implement it in different scenarios and environments to reach optimum performance.

This paper is organized as follows: Section II introduces the commonly related works that proposed congestion control approaches. Section III presents the concept of the SCReAM congestion control algorithm and its design. Section IV highlights the evaluation environment, measurement setup, and the used datasets in our experiments. Section V presents the results obtained from our experiments including early-stage and 5G networks followed by a detailed discussion. Finally, future directions and the conclusion are discussed in Section VI.

II. RELATED WORKS

In this section, we will discuss how congestion control is employed and the related congestion control algorithms, starting from the early TCP version, TCP variants, and the most popular congestion control algorithms. Congestion control can be employed in two main scenarios: to reduce network congestion when it is detected, or prevent it from happening in the first place. There are distinct parameters that can be chosen as the backbone of the congestion control algorithm which is used in the feedback process to report the network state. Also, congestion control algorithms can be classified using many metrics such as fairness criterion uses, aspect of performance, network deployment ability, and feedback type & size. Therefore, it is important to employ a congestion control algorithm that has the ability to control and eliminate any causes of congestion. Congestion control algorithms are being used as a feedback system that reports the performance of the network every certain time interval, which is the reason that it is considered the largest feedback system that has been deployed artificially nowadays [7].

Participation in academic research in this area is also increasing due to its ability to extend and offer new services by proposing different approaches to address the requirements of the targeted environment. Most of the current research aims to address the most common issue; congestion control. Many people believe that due to the increasing bandwidth limitation in computer networks, network congestion is no longer an issue [8]. However, it is still an issue due to the emerging technologies that are in constant change, thus, it demands higher network requirements.

Early design of TCP [9], [10] included a simple approach without extensive performance that targets the congestion issue itself, go-back-n is used where packets are sent without waiting for feedback or ACKs. ACKs are sent by the server when all packets arrive in order and are error-free. Further improvements are done as follows:

- The slow start approach has been implemented in various TCP schemes using a variable called congestion window (CWND); it defines the number of packets that can be sent during a certain time interval.
- Another parameter “ssthresh” has been imported into further designs in the congestion avoidance phase. This parameter declares the appropriate CWND size based on network load.
- Round Trip Time (RTT) is used to define a specific period where the packets should reach the destination, if RTT is expired, packets are retransmitted again.
- Duplicate Acknowledgments (ACKs) are used when the receiver receives out-of-order segments. Fast retransmit is used to send these segments without waiting for the timer to expire.

The first TCP implementation that included the previously mentioned improvements is TCP Tahoe. Another implementation of TCP referred to as “TCP Reno” (4.3BSD) included a new algorithm called the fast recovery. In this version, duplicate ACKs mean the packets have reached successfully, when the sender receives 3 duplicate ACKs, it means that a segment is lost, this algorithm enters the fast recovery mode, and the sender sends the lost segment, CWND is reduced by half, ssthresh is updated. When multiple segments are lost, TCP Reno exits the fast recovery mode when receiving partial ACKs which results in a timeout.

TCP NewReno defined in RFC 6582 [11] is different from the last version in the part of receiving partial ACKs, TCP NewReno does not exit the fast recovery mode, instead, it assumes that the most recent acknowledged segment is already delivered, it only retransmits the segment that follows that one. Thus, it eliminates the retransmission timeouts that the algorithm should wait until it sends another segment. Fast retransmission starts when three ACKs are received and ends when all up-to-date segments are acknowledged or when a retransmission timeout occurs. TCP Vegas [12] is an older version than NewReno. It estimates the expected throughput in the network, then compares the actual throughput to the expected throughput, if the latter is higher, there is congestion in the network. Hence, CWND must be modified, for example, in a congestion state, CWND is reduced to offer the network more time to balance the load. Another important fact in Vegas, if the timeout value is lower than the RTT estimate, it requires only one ACK to retransmit the segment. Further information about TCP variants can be found in [13], [14], and performance analysis is done in [15], [16].

A new congestion control algorithm “D-TCP” [17] is proposed for mmWave NR. The adaptive increase adaptive decrease scheme is used to control the congestion window and bandwidth of the networking by estimating the accessible amount that can be utilized. In high-BDP, the algorithm is able to fully utilize the available bandwidth while reducing packet loss to a minimum. Real-time live air and simulations were used in experiments to evaluate this algorithm in the LTE network. This algorithm achieves higher goodput than TCP-Reno and TCP-Cubic.

SCTP congestion control is enhanced over LTE-A “ENH-SCTP” [18] to achieve effective SCTP congestion. To improve the performance of SCTP, congestion avoidance and slow start have been used based on a new congestion window approach called multi-criteria decision-making (MCDM). This approach utilizes various network metrics such as queue size, number of lost packets, number of received packets, number of sent packets, throughput, and CWND. The value of each parameter is chosen as the optimum value that will result in the best performance based on an algorithm called Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). Based on experiments done through simulations, the proposed algorithm resulted in a better performance than TCP and STD-SCTP.

Google congestion control [19], [20] is an algorithm designed to address WebRTC requirements. One-way delay variation is used in an end-to-end manner which is estimated using the Kalman filter. The sending rate is dynamically throttled by an adaptive threshold that will be compared to the estimated value. Since Chrome WebRTC stack and Google hangouts are using RTP/RTCP, this algorithm has been implemented over RTP/RTCP too. Google Chromium browser was used to obtain results that indicated that this algorithm is able to track link capacity by adapting the sending rate, reverse traffic does not imply any effect on this algorithm, and with short & long-lived TCP flows, it offers fairness for inter-protocol and intra-protocol. It is still a challenge for this algorithm to react fast enough to congestion while providing high throughput because although it uses delay and loss as the metrics to adjust the rate, the sending rate is being adjusted by a single mechanism. Further improvement to this algorithm is illustrated in [21].

Finally, SCReAM [5], [6] is a rate adaptation congestion control algorithm that combines two types of algorithms (delay-based and loss-based) to create a hybrid model for LTE networks. This algorithm utilizes the concept of the packet conservation principle to prevent the network from getting congested. Furthermore, the authors described an overview of the proposed model followed by a realization of congestion control feedback and the estimation of CWND. Since SCReAM offers many advantages over other algorithms and it resulted in a better performance compared to the rate-based algorithm, we decided to choose SCReAM as the base for further experiments and evaluation. A detailed description of this algorithm is discussed in the next section.

III. SCREAM PROTOCOL AND ITS PARAMETERS

A. SCReAM Protocol

Self-Clocked Rate Adaptation for Multimedia (SCReAM) [5], [6] is a congestion control algorithm that adapts to the changes in the network to estimate different variables such as rate, and congestion window, queuing delay, etc. Based on the estimations, it adapts to such changes by modifying its network parameters to result in optimum performance. In this study, we selected SCReAM over rate-based algorithms to perform our evaluations due to many reasons; it reduces

the variations of short-term delay due to a more efficient congestion window computation method, and it requires a shorter time scale operation due to the self-clocking feature. The issues of the rate-based algorithm mentioned before can be addressed efficiently by SCReAM.

There are multiple concepts revolving around SCReAM which are having similarities that are worth mentioning (for a detailed description of the difference between the actual concepts in the previously proposed schemes and the implemented concept inside SCReAM, refer to [5]):

- A Congestion control concept that is window-based and TCP-friendly in which the self-clocking concept was previously used [22].
- Packet conservation principle [23] is supported by SCReAM.
- Congestion is implemented over RTP streams [24].
- Designed to support WebRTC [25].
- SCReAM follows a similar manner to LEDBAT [26] when calculating the congestion window.
- Queue delay is also measured similarly to LEDBAT [26].
- Reduced size RTCP is used as feedback based on [27].

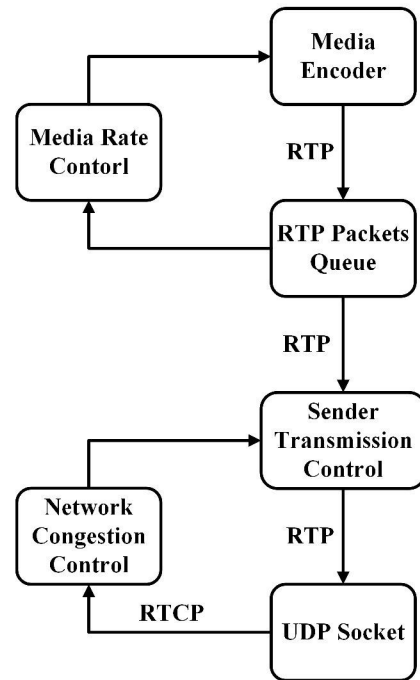


Fig. 1: SCReAM Components [6].

As shown in Fig.1, the main components of SCReAM (located at the sender side) are media rate control, sender transmission control, and network congestion control. Other components include Queue RTP packets and UDP sockets. The receiver side is only used to generate feedback. The process of adjusting media bitrate (target rate) is handled by media rate control. Adjusting bitrate refers to either increase or decrease based on a threshold that is compared to RTP queue size. To determine the transmitted data, the congestion window must be related to the bytes in flight, which is done in

the sender transmission control. The procedure of transmitting data is affected by other parameters such as estimated link throughput and packet size. Bytes in flight refers to the amount of data inside the network at a certain interval, it is determined using The network congestion control. The feedback from the receiver is used to specify congestion window size.

SCReAM components are performing the following tasks from start to end: when media frames are received, the media encode starts the encoding process, then, the encoded media are sent to the RTP queue. Later, the rate adaptation starts based on the RTP queue size to specify the **target bitrate**. Furthermore, RTP packets are selected to be sent to the UDP socket. The sender transmission control and network congestion control exchange the necessary information after RTCP packets are received.

B. SCReAM Parameters

The complicated design of SCReAM has led to a complexity in the number of parameters used for precise decision-making when deciding the values that directly affect SCReAM's performance. Only certain parameters are mentioned in this study, for example, more than 20 constants and 20 state variables can be found in RFC8298 [6]. However, due to the complex coding design, much more parameters can be found in [28]. When creating a new stream, certain Parameters must be included as shown in Table I. In this study, we will focus on specific parameters that we will discuss in the following section.

In order to identify two important input and output parameters; frames and bitrate, we created a flowchart which includes a detailed description of each step in the SCReAM algorithm, the flowchart is shown in Fig.2. It is important to note that the trace video file represents the input frames, **bytesRtp** labeled in red color represents the output value for each frame and **TotalBytes** parameter represents the output values for all frames. The **Trace_Video** represents the input file that will be used in the video encoder, further information is discussed in section VI. Based on Fig.2, Equation 1 follows step 3, **Scale_Factor** represents the percentage between **target_Bitrate** and **nominalBitrate** where **targetBitrate** is the expected bitrate for the output and **nominalBitrate** is the video encoder bitrate. Then, **Video_Encoder_bytes** is realized by multiplying the **frameSize** by the **Scale_Factor** as shown in Equation 2. Finally, as illustrated in Equation 3, **nominalBitrate** is updated everytime based on its **current value**, **frameSize**, and **frameRate**.

$$\text{Scale_Factor} = \frac{\text{targetBitrate}}{\text{nominalBitrate}}; \quad (1)$$

$$\text{Video_Encoder_bytes} = \text{frameSize} * \text{Scale_Factor} \quad (2)$$

$$\text{nominalBitrate} = 0.95 * \text{nominalBitrate} + 0.05 * \text{frameSize}[ix] * \text{frameRate} * 8 \quad (3)$$

TABLE I: Stream Registration Parameters

Parameter	Initial Value
rtpQueue	rtpQueue[0]
ssrc (Synchronization Source)	10
priority	0.7 f
minBitrate	1 Mbps
startBitrate	1 Mbps
maxBitrate	10 Mbps
rampUpSpeed	10 Mbps
rampUpScale	1.0 f
maxRtpQueueDelay	0.2 f
txQueueSize Factor	0.2 f
queueDelayGuard	0.1 f
lossEventRateScale	0.9 f
ecnCeEventRateScale	0.95 f
isAdaptiveTargetRateScale	True

IV. MEASUREMENT SETUP AND DATASETS

The SCReAM algorithm can be implemented in two different methods as stated in [28]. The first method is by using a Windows-based test application through the Visual Studio software, which we are currently using. The other method is by using Linux based BW test application, which we used previously to ensure that the SCReAM is working as expected. SCReAM BW test application can be built and worked on Ubuntu 16.04 and later [28]. The method that we are currently using to implement the algorithm consists of multiple c++ codes including **sender**, **receiver**, supportive classes (**RTP Queue**, **Net Queue**, and **Video Encoder**), and the coordinator code (**scream_v_a**) which creates the environment and controls all of the other codes. The RTP Queue class is used for the Rudimentary RTP queue while the Net Queue class is used for the Simple delay and bandwidth limitation. In addition, the video encoder class is used as a Simple model of a video encoder.

For the tracing, we used the video model trace file (2-70KB) [28] as input frames. As shown in Fig.3, we print the input file from the Video Encoder part to ensure that the frames are properly inserted into the algorithm. The total size of the file is 61.8 MBps (494.4 Mbps). After encoding, the input file will be multiplied by a scale factor + RTP overhead. Note that the time required to read one video file is equal to 47.74 seconds which is the total number of frames (2387) divided by the Frame Rate (FR) which was set in the algorithm as 50 Frames Per Second (FPS).

We tested the SCReAM algorithm performance in the 5G Network by using the 5G dataset traces, which are collected over a 60GHz WLAN testbed hosted at the University of New York NYU [29], shown in Fig.4. The bitrate range of the 5G dataset is between 0 - 3080 Mbps. We used the 5G dataset to control the available bandwidth of the data transmission to emulate the 5G environment. In this study, we have used 5G datasets that represent measurements of dynamically changing

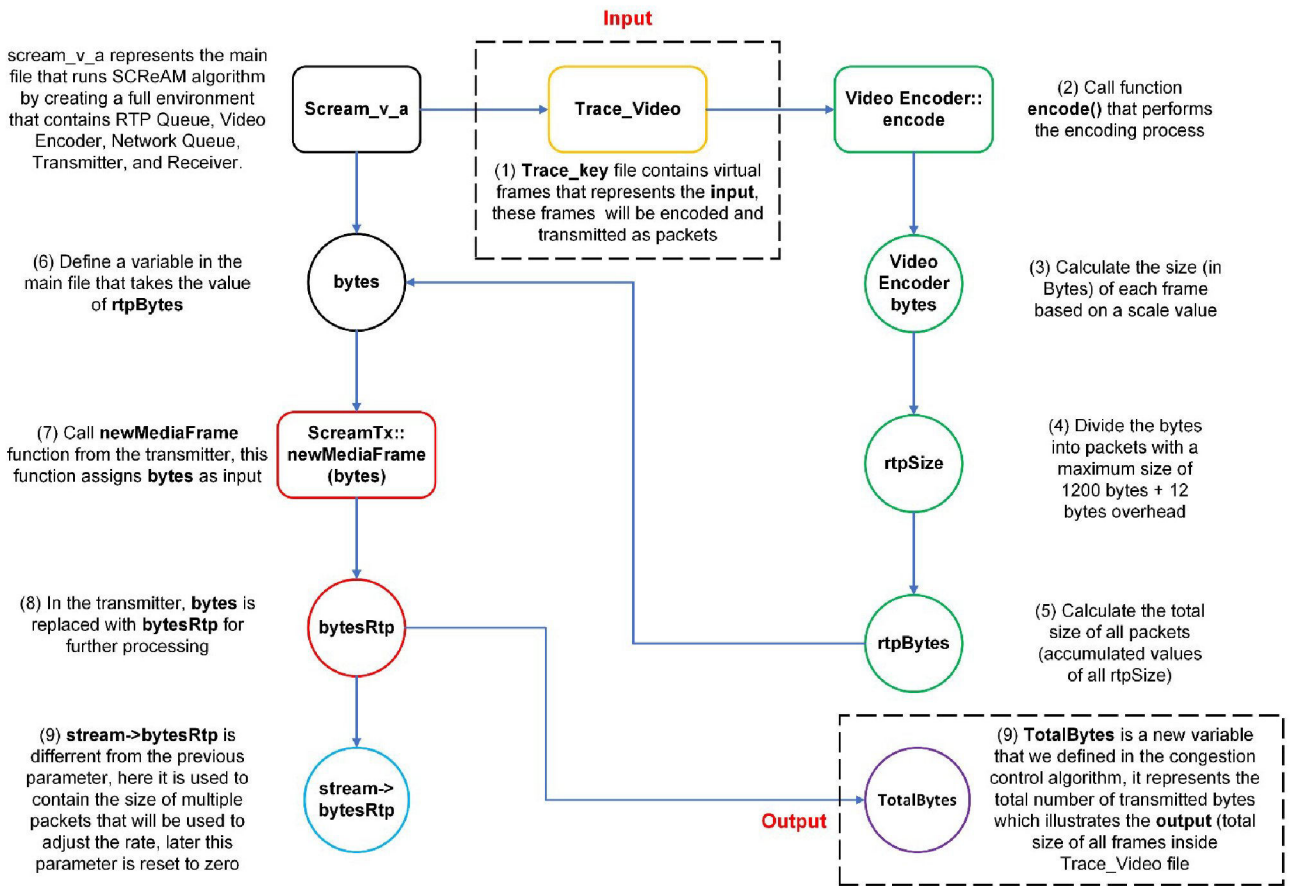


Fig. 2: Flowchart of the Input/Output Frame Size.

network bandwidth (Capacity) that have been extracted from real-time 5G measurements. Hence, it characterizes as a real 5G environment because we are using the extracted data as input in our experiments. Thus, we are emulating a 5G environment.

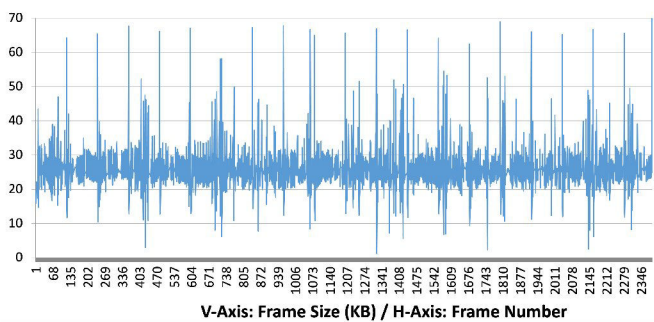


Fig. 3: Original Video Trace.

V. RESULTS

In this section, we will introduce two experiments that aim to test the performance of the SCReAM congestion control algorithm in order to identify the input and output regarding video frames and bitrate. These two experiments

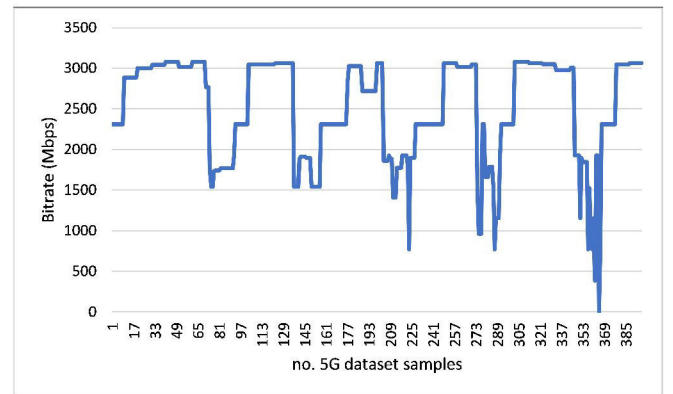


Fig. 4: 5G Dataset Traces Used as Network Shaping.

are realized to illustrate the difference between the default and our modified (using 5G dataset) implementations. Several metrics were investigated to illustrate our results properly. The following metrics were selected for our experiments: **frame size**, **target bitrate**, **throughput (rateTransmitted)**, **RTP queue delay**, **network queue delay**, **bytes in flight**, and **congestion window (CWND)**. These metrics were chosen due to their importance in presenting the effect of any modification to the algorithm.

In the first experiment, the available bandwidth of the algorithm is fully controlled by a step function (throttling function) divided into two intervals with a total simulation time of 100 seconds. For the second experiment, the available bandwidth of the algorithm is controlled by a step function in the first interval, and it is controlled by the 5G dataset in the second interval for the same simulation time. The first interval is from 0 - 47.73 seconds and 95.49 - 100 seconds, while the second interval is from 47.74 - 95.48 seconds. We divided the simulation time into two intervals (assuming that the third interval of 95.48 - 100 seconds does not pose a considerable effect on our results), each with a time of 47.74 seconds represents the time length required to read one video trace file in order to show the impacts of the various throttling function values on the encoded video file.

In the first experiment, the encoded video trace file of the algorithm is represented in Fig.5. It is worth mentioning that the slow start that follows the target bitrate and rateTransmitted respectively is important in the SCReAM algorithm itself to avoid congestion in the network. However, it does create a certain loss in the first 200 frames. In addition, there is a slight difference deviated from the actual value of the algorithm throughput (rateTransmitted) due to the adaptation effect. Target bitrate and rateTransmitted are presented in Fig.6. Hence, the target bitrate (the expected throughput) directly affects the scale value resulting in changing the actual size of the frame.

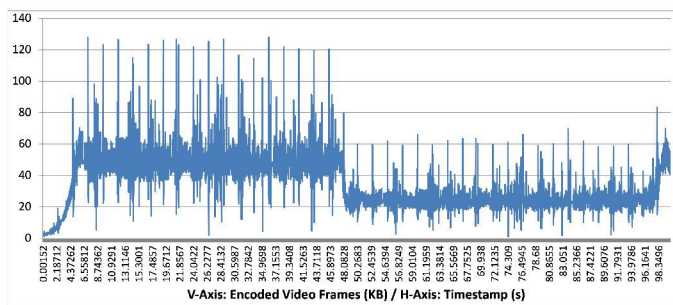


Fig. 5: Encoded Video Trace.

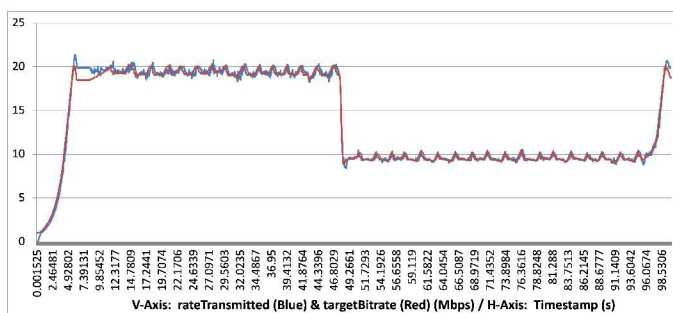


Fig. 6: Target Bitrate and Throughput (rateTransmitted).

It is very important to mention that the algorithm is designed to continue transmitting the video file until the end of the simulation time, which is set to 100 seconds. The output for the second interval is not necessarily the same because the

target bitrate is different at each time interval which implies that the calculations within this algorithm are constantly done to alter the values of different parameters to result in a less congested network.

To identify that the algorithm is performing properly, we set up the algorithm with a specific throttling value that is used for the network shaping to limit the output throughput. All experiments done in this phase are represented in 100 seconds which are divided into two intervals. The throttling value for the first interval was 20Mbps while the throttling value for the second interval was 10Mbps. We also modified the number of output values to simplify the results for proper illustration (hence that over 100 seconds, it is possible to result with 100 or 100000 values based on the specified time interval). Fig.7 shows the value of SCReAM initial network rate (Blue), which represents the throttling values for the two intervals, and the SCReAM throughput (rateTransmitted) (Red).

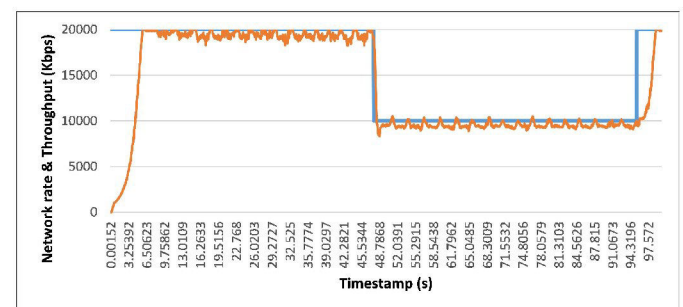


Fig. 7: Initial Network Rate and Throughput (rateTransmitted).

The RTP queue delay (Blue) and network queue delay (Red) are shown in Fig.8. RTP queue delay is related to the video frame delay through the network node. The simulation results of RTP queue delay were between 0 and 101 ms. The network Queue delay is the time needed for a network request to go between the sender and the receiver back and forth. The simulation results of network queue delay were between 0 and 76 ms.

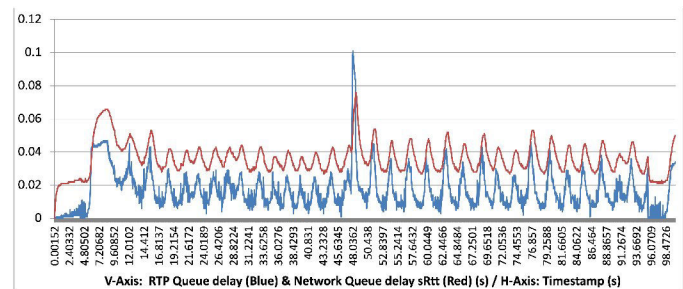


Fig. 8: RTP Queue Delay and Network Queue Delay (sRtt).

Finally, Fig.9 shows the bytes in flight (Red) and congestion window CWND (Blue). The term “bytes in flight” represents how many bytes can be in the network while CWND is used by the transmission scheduler to calculate the number of bytes to send into the network. The initial value of CWND is set

to 15 KB (default value). The values range of the Bytes in Flight for the interval of using the throttling value of 10 Mbps is between 32 - 150 KB, while this range for the CWND is between 95 - 150 KB.

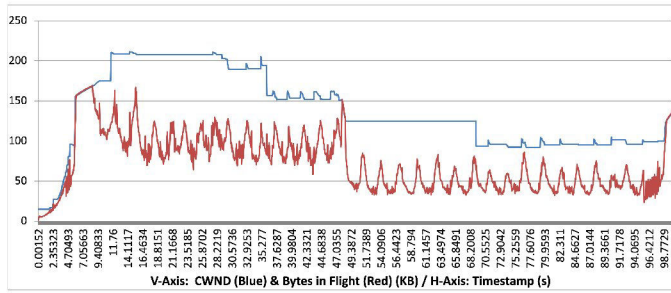


Fig. 9: CWND and Bytes in Flight.

The second experiment is done to evaluate and analyze the performance of the SCReAM algorithm in 5G Networks by replacing the fixed throttling value (for a certain interval) with the bitrate values from the 5G dataset Fig.4. We successfully modified the algorithm code to be able to read the bitrate from the 5G dataset file, which has a bitrate range of 0 - 3080 Mbps. We modified the default values of some parameters including the initial network rate, MaxBitrate. Since the algorithm's maximum bitrate is 100Mbps, we set the value of the MaxBitrate and the initial rate to 100Mbps to get the optimal performance. Regarding the throttling values, we maintained the throttling value of 20 Mbps in the first interval, while changing the second interval's throttling value to the 5G dataset's values. The video trace file used in both experiments is the same because it is important to monitor and analyze the changes in SCReAM's performance when the 5G dataset is applied.

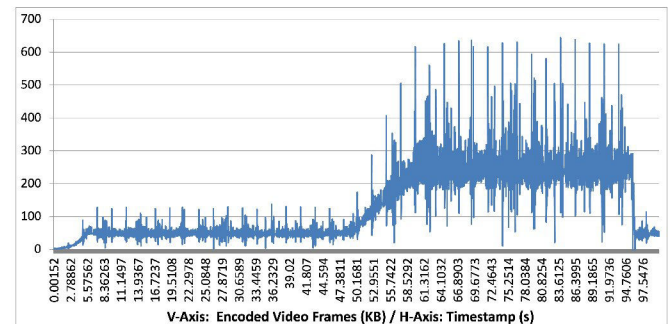


Fig. 10: Encoded Video Trace with 5G Dataset.

Fig.10 illustrates the impact of using the 5G dataset in the encoded trace video file. The reason behind such changes in the encoded frames' sizes is due to the higher value of the target bitrate that is used as a scale value to calculate each frame size. The frames' size of the second interval (from 47.74 - 95.48 seconds) is almost five times the frames' size of the first interval when we used a fixed throttling value of 20 Mbps. It happened because the bitrate of the algorithm is limited to approximately 100 Mbps, which is the maximum bandwidth

limitation of the SCReAM. For instance, a frame size of 100 KB will become ≈ 500 KB when the 5G dataset is applied. This implies that the algorithm mimics the encoding at higher resolution (better video quality) at the higher bitrates.

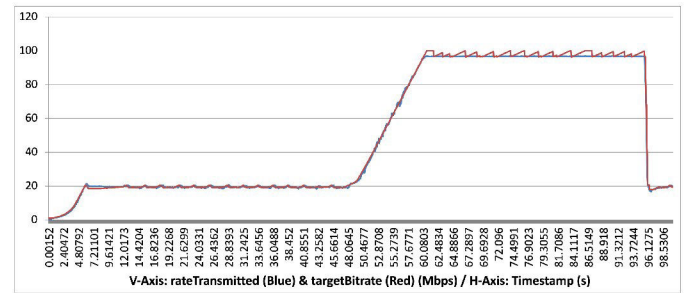


Fig. 11: Target Bitrate and Throughput (ratetransmitted) with 5G Dataset.

The target bitrate; the expected bitrate of the output (Blue) and the rateTransmitted; the actual bitrate of the output (Red) when using the 5G dataset are shown in Fig.11. The interval of using the 5G dataset (from 47.74 - 95.48 seconds) has a maximum bitrate of 96.85Mbps, which approximately equals the maximum algorithm bitrate 100Mbps. As previously mentioned, The target bitrate can be also defined as the desired bitrate and it is related to the media rate control. After the video frames are encoded to RTP packets and then pushed into the RTP queue, the RTP queue length is reported to the media rate control unit which calculates the optimal target bitrate and then feeds it to the video encoder to regulate its bitrate.

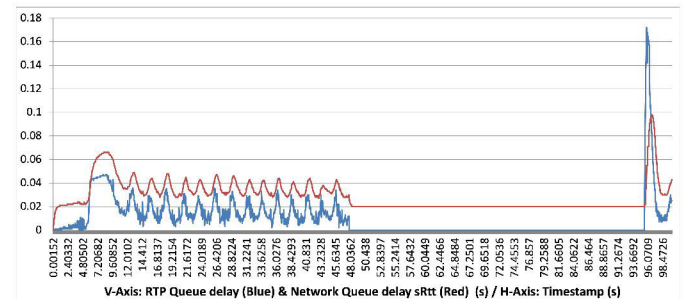


Fig. 12: RTP Queue Delay vs Network Queue Delay (sRtt) with 5G Dataset.

Fig.12 presents the RTP queue delay (Blue) and Network Queue delay (Red) when the 5G dataset is used. The values range of the RTP queue delay in the first interval is between 0 and 172 ms, while this range is between 20 and 30ms when the 5G dataset is used. On the other hand, the values range of the network queue delay is between 0 and 98 ms in the first interval, while it is between 0 and ≈ 20 ms when the 5G dataset is applied.

Finally, Fig.13 depicts the bytes in flight and the congestion window. The following results illustrated the second interval only (when the 5G dataset is applied). The range of values

for bytes in flight is 46 - 245 KB, while the range of the congestion window is 163 - 306 KB.

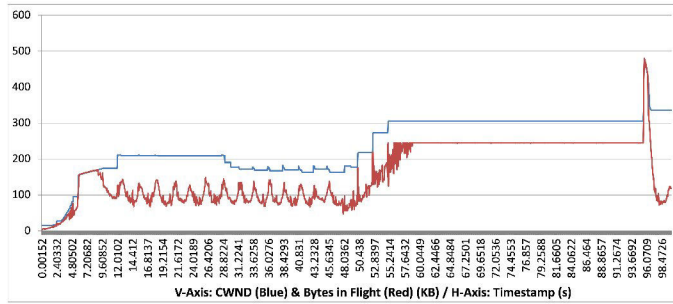


Fig. 13: CWND and Bytes in Flight with 5G Dataset.

VI. CONCLUSION

In this study, we aim to explore the possibility of achieving optimal streaming in a 5G environment. We are especially interested in choosing the appropriate congestion control algorithm to achieve this purpose. As a candidate, we are focusing on the Self-Clocked Rate Adaptation for Multimedia (SCReAM) congestion control algorithm. Also, we aim to provide guidelines for achieving the video streaming performance of SCReAM in 5G networks. We discussed the design of SCReAM and its parameters followed by the measurement setup and the used datasets. Finally, we evaluated the performance of SCReAM based on two scenarios; a step-function controlled available bandwidth scenario with default settings and the 5G dataset which is used as a throttling function to specify the available bandwidth for the algorithm.

Many parameters were investigated with and without throttling in the default settings and the 5G dataset such as frame size, target bitrate, throughput (rateTransmitted), RTP queue delay, network queue delay, bytes in flight, and congestion window (CWND). Our work supports our further research plan to implement it in different scenarios such as [30], environments, and parameters to reach optimum performance. Furthermore, since we modified the algorithm to be used in 5G-like networks, one of the future goals is to test this algorithm in a dedicated 5G scenario after making major modifications to it.

REFERENCES

- [1] A. Kumar, P. Srinivas, and A. Govardhan, "A review on congestion control approaches for real-time streaming application on the internet," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 16, no. 4, pp. 23–28, 2018.
- [2] V. Kushwaha and R. Gupta, "Congestion control for high-speed wired network: A systematic literature review," *Journal of Network and Computer Applications*, vol. 45, pp. 62–78, 2014.
- [3] B. Subramani and E. Chandra, "A survey on congestion control," *Global Journal of Computer Science and Technology*, 2010.
- [4] L. De Cicco, G. Carlucci, and S. Mascolo, "Congestion control for webrtc: Standardization status and open issues," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 22–27, 2017.
- [5] I. Johansson, "Self-clocked rate adaptation for conversational video in lte," in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, 2014, pp. 51–56.
- [6] I. Johansson and Z. Sarker, "Self-clocked rate adaptation for multimedia," Tech. Rep., 2017.

- [7] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," *IEEE control systems magazine*, vol. 22, no. 1, pp. 28–43, 2002.
- [8] R. Jain and K. Ramakrishnan, "Congestion avoidance in computer networks with a connectionless network layer, part i: Concepts, goals and methodology," *arXiv preprint cs/9809095*, 1998.
- [9] M. Allman, V. Paxson, and E. Blanton, "Tcp congestion control," Tech. Rep., 2009.
- [10] M. Allman, V. Paxson, and W. Stevens, "Rfc2581: Tcp congestion control," 1999.
- [11] S. Floyd, T. Henderson, and A. Gurtov, "The newreno modification to tcp's fast recovery algorithm," Tech. Rep., 2004.
- [12] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *Proceedings of the conference on Communications architectures, protocols and applications*, 1994, pp. 24–35.
- [13] S. Fahmy and T. P. Karwa, "Tcp congestion control: overview and survey of ongoing research," 2001.
- [14] G. A. Abed, M. Ismail, and K. Jumari, "A survey on performance of congestion control mechanisms for standard tcp versions," *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 12, pp. 1345–1352, 2011.
- [15] O. Ait-Hellal and E. Altman, "Analysis of tcp vegas and tcp reno," *Telecommunication systems*, vol. 15, no. 3, pp. 381–404, 2000.
- [16] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of tcp reno and vegas," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 3. IEEE, 1999, pp. 1556–1563.
- [17] M. R. Kanagarathnam, S. Singh, I. Sandeep, H. Kim, M. K. Maheshwari, J. Hwang, A. Roy, and N. Saxena, "Nexgen d-tcp: Next generation dynamic tcp congestion control algorithm," *IEEE Access*, vol. 8, pp. 164 482–164 496, 2020.
- [18] I. A. Najm, M. Ismail, J. Lloret, K. Z. Ghafoor, B. Zaidan, and A. A.-r. T. Rahem, "Improvement of sctp congestion control in the lte-a network," *Journal of Network and Computer Applications*, vol. 58, pp. 119–129, 2015.
- [19] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (webrtc)," in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, pp. 1–12.
- [20] G. Carlucci, L. De Cicco, and S. Holmer, "Congestion control for web real-time communication," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2629–2642, 2017.
- [21] L. De Cicco, G. Carlucci, and S. Mascolo, "Understanding the dynamic behaviour of the google congestion control for rtwebr," in *2013 20th International Packet Video Workshop*. IEEE, 2013, pp. 1–8.
- [22] S. H. Choi and M. Handley, "Fairer tcp-friendly congestion control protocol for multimedia streaming applications," in *Proceedings of the 2007 ACM CoNEXT conference*, 2007, pp. 1–2.
- [23] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.
- [24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," Tech. Rep., 2003.
- [25] C. Holmberg, S. Hakansson, and G. Eriksson, "Web real-time communication use cases and requirements," *Request for Comments (RFC)*, vol. 7478, 2015.
- [26] S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind *et al.*, "Low extra delay background transport (ledbat)," in *RFC 6817*, 2012.
- [27] I. Johansson, M. Westerlund *et al.*, "Support for reduced-size real-time transport control protocol (rtcp): Opportunities and consequences," RFC 5506, April, Tech. Rep., 2009.
- [28] Ericsson-Research, "Scream - mobile optimised congestion control algorithm." [Online]. Available: <https://github.com/EricssonResearch/scream>
- [29] S.-H. GitHub. Traces of link capacity collected over a 60ghz wlan testbed hosted at nyu wireless. [Online]. Available: <https://gist.github.com/Shreeshail-Hingane>
- [30] S. Zhang, W. Lei, W. Zhang, and Y. Guan, "Congestion control for rtp media: A comparison on simulated environment," in *International Conference on Simulation Tools and Techniques*. Springer, 2019, pp. 43–52.