Performance Evaluation of L4S in XR Scenarios

Philipp Steininger*, Rastin Pries†, Yash Deshpande‡, Kaan Aykurt‡, Chia-Yu Chang†,
Koen De Schepper†, Wolfgang Kellerer‡
*C4B Com For Business AG, Germany
† Nokia Strategy&Technology, Nokia, Germany and Belgium
‡Chair of Communication Networks, Technical University of Munich, Germany
Email: ph.steininger@yahoo.com, rastin.pries@nokia.com, {chia-yu.chang, koen.de_schepper}@nokia-bell-labs.com
{yash.deshpande, kaan.aykurt, wolfgang.kellerer}@tum.de

***Abstract*—Low Latency, Low Loss, Scalable Throughput (L4S) is a network protocol designed to provide ultra-low queuing delays, minimal packet loss, and scalable throughput, which are key factors for real-time applications such as streaming, online gaming, and Extended Reality (XR). With the rise of advanced XR applications and the increasing adoption of various XR headsets, remote rendering has become a common practice due to the hardware limitations of these devices.**

This study evaluates the performance of L4S in multi-device XR scenarios, focusing on its impact on latency, packet loss, and throughput. Unlike traditional congestion control mechanisms, which rely on reactive loss-based methods, L4S employs Explicit Congestion Notification (ECN) to signal congestion along with fast congestion control algorithms to minimize packet loss.

This proactive approach enables rapid adaptation to network conditions, ensuring consistently low latency and improved stability. L4S was integrated into the network infrastructure between a streaming application and an XR headset. The performance was tested under varying network conditions to assess its effectiveness. The results show that L4S significantly reduces packet loss and latency while maintaining high throughput, leading to enhanced XR streaming quality and real-time interactions. These findings demonstrate the potential of L4S to improve real-world XR applications, with implications for broader adoption in low-latency networking.

***Index Terms*—L4S, low latency, XR**

I. Introduction

Extended Reality (XR) applications, encompassing Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR), are transforming industries such as logistics, healthcare, education, and entertainment by providing immersive and interactive experiences. However, these applications demand substantial computational power to render high-quality graphics and process real-time interactions, often exceeding the capabilities of mobile and wearable devices. To address this limitation, remote rendering techniques are employed to offload computationally intensive tasks to edge servers, reducing the processing burden on client devices while maintaining a realistic user environment [1]–[3]. This approach allows to develop lightweight XR devices with extended battery life and improved performance. However, remote rendering also induces network bandwidth, latency, and reliability challenges [4]. The network infrastructure must support real-time data transmission between XR clients and edge servers, ensuring seamless and responsive interactions [5]. The quality of streaming applications is susceptible to network performance [6], [7]. High latency can cause noticeable delays, reducing the immersive experience and interactive sessions. Packet loss can result in video and audio glitches, disrupting the viewing experience. On the other hand, high-throughput is necessary to support high-definition video streams, especially as content resolutions continue to increase.

Latency in a network is primarily caused by congestion from other traffic competing for the same links, which leads to queuing delays as packets wait to be processed at intermediate network nodes such as routers or switches [8]. When the rate of incoming traffic exceeds the processing capacity of a router or switch, packets accumulate in buffers, increasing the delay. If the queue reaches its maximum capacity, additional incoming packets are dropped, resulting in packet loss and requiring retransmissions that further exacerbate latency. Effective congestion control mechanisms, such as active queue management and traffic shaping, are essential to reduce latency and mitigate packet loss to improve the overall Quality of Experience (QoE) for remote rendering of XR applications.

Low Latency, Low Loss, Scalable Throughput (L4S) [9] is a promising solution for effective congestion control, enabling ultra-low queuing delays while maintaining high throughput. L4S supports scalable congestion control algorithms that react swiftly to network conditions, reducing latency even under high traffic loads. This paper evaluates L4S on real remote rendering applications and with real devices, providing insights into its capabilities and utility. First, it evaluates the bandwidth requirements and latency headroom, for example, in a scene with a high-resolution 3D car model. Then, it compares the performance of L4S against classical flows when the available bandwidth to the flow changes. Results show a far superior performance of L4S compared to classical flows regarding the two main KPIs important for XR applications: latency and packet loss.

Philipp Steininger's work on this paper was conducted while employed at Nokia.
ISBN 978-3-903176-72-0 ©2025 IFIP

II. Background

A. Low Latency, Low Loss, Scalable Throughput (L4S)

Classical rate control mechanisms rely on packet loss notifications to adjust the sending rate, requiring frequent losses in dynamic network conditions. This approach is not suitable for real-time applications, as packet loss and queue build-up induce noticeable QoE degradation. L4S addresses this limitation by proactively detecting and indicating bottleneck conditions and adjusting the rate before losses occur, theoretically outperforming classical methods.

L4S flows are identified by flagging the Explicit Congestion Notification (ECN) bits (the two least significant bits of the traffic class section of an IP header). This allows L4S flows to be backward compatible since all existing routers that do not support this procedure ignore the ECN flag. RFC 3168 [10] specifies that the two unused bits inside the original TOS field of the IP header will be used as ECN bits. To mark packets compatible with L4S, a sender has to mark the packet with the ECT(1) flag. Network nodes identify the existence of this flag and modify the Congestion Experienced (CE) bit according to the link utilization. For example, if a router's bandwidth is used to 50%, then half of the packets will have the CE bit set. When the packets arrive at the receiver, they are processed to extract the bits and the information is kept in two counters. One for received packets marked with ECT(1) and one for packets marked with CE. At a predefined fixed interval (for example the frame rate of a video), these counters are transmitted to the sender via a Congestion Notification Report and reset. The sender then adjusts its sending rate according to the reported congestion.

A detailed overview of how network nodes should perform packet marking can be found in RFC9332 [11]. L4S-compatible network nodes work by using different queuing mechanisms for different flows. A simplified illustration can be seen in Fig. 1, where the node must handle classical and L4S-compatible flows. First, a classifier checks incoming packets for the existence of the L4S flag and puts the packet in its respective queue. This separation ensures that flows that maximize the buffer (e.g., TCP Reno) do not negatively affect the latency of the L4S flow. The L4S processor is responsible for marking the CE and ECT(1) flags of the packet headers. A priority scheduler is used to determine which packet is forwarded at the respective port to ensure that both flows receive a fair share of the network bandwidth.

When the sender receives the Congestion Notification Report, it adjusts its bitrate accordingly. RTPrague [12] is used along with L4S, which combines the congestion percentage and measured Round Trip Time (RTT). Thus, it can be seen that the full benefits of L4S are achieved only when compatible protocols—such as scalable congestion control, accurate Explicit Congestion Notification (ECN), and flow isolation at queues—are implemented at both endpoints of the connection as well as at the bottleneck router. Moreover,

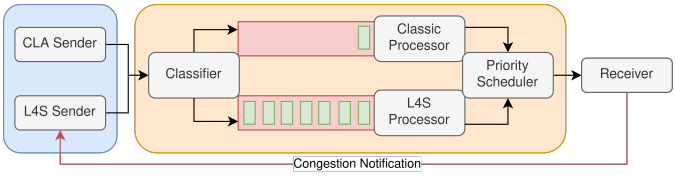

Fig. 1: L4S Router Port Egress: The router maintains separate queues for L4S and Classical (CLA) flows. A classifier first checks the flows and puts the packets in the appropriate queue. Then, L4S packets are marked according to RFC9332 by the L4S processor. A priority scheduler at the egress finally decides which packet to forward and ensures that both queues get a fair share of the link.

a real XR application must be deployed with a compatible encoder for a realistic QoE assessment, and relevant metrics must be collected while changing the network conditions. The implementation of L4S in programmable switches was evaluated in [13], and the results reported that RTPrague performs better than Cubic as a congestion control algorithm. However, only the reaction of the system to a reduction in bandwith was considered, and the latency measurements were not reported in this work. To the best of our knowledge, this is the first paper evaluating L4S on a real XR device.

III. Measurement Setup

The measurement setup comprises four computers, including three Linux PCs and one Windows PC functioning as an XR streaming server. The streaming server has an AMD Ryzen 7950X processor, an NVIDIA RTX 4090 GPU, and 64 GB RAM to process high loads for XR rendering. The network infrastructure includes two switches and a wireless Access Point (AP), which is for some experiments a complete 5G network with a Nokia base station and core network and for other experiments a WiFi AP. The experimental client is a Microsoft HoloLens 2 with a Qualcomm Snapdragon 850 Compute Platform. Additionally, two Linux PCs generate background traffic to simulate congestion on the link between the router and the switch. A detailed schematic of the test configuration is presented in Fig. 2.

The scenes use Unity, a 3D game engine with various XR features. A custom streaming application running at the client and server acts as an interface between Unity and WebRTC. The client on the HMD gathers spatial position and orientation with six degrees of freedom and controller and/or hand-tracking information. This data is packetized and sent to the server. The server processes this data and renders a frame on the server's GPU. This frame is then packetized and paced over the network at a rate determined by the congestion control algorithm. After all packets have arrived, the frame is successfully assembled and decoded to be realigned to the current spatial location [3] and displayed on the HMD.

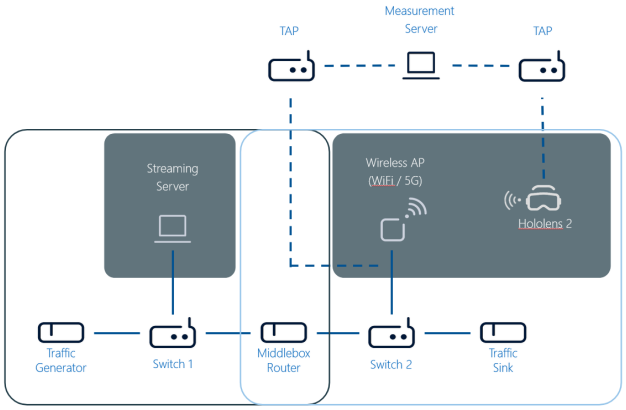

Fig. 2: Measurement Setup: The streaming server renders the scenes and streams them to the Hololens 2 via a middlebox router capable of L4S. A traffic generator and sink creates congestion on the shared link.

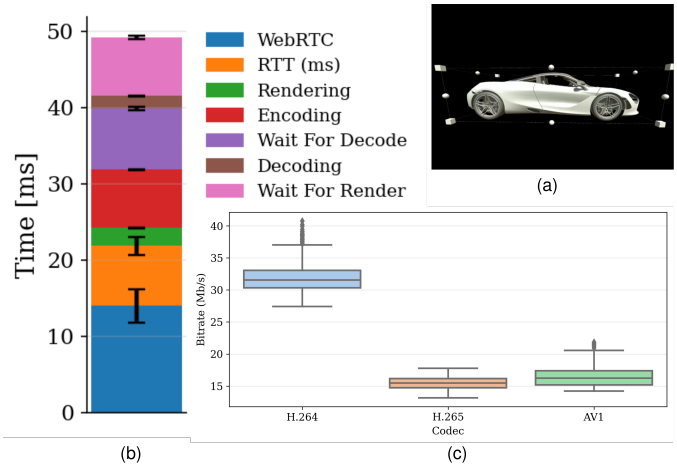

Fig. 3: (a) A simple McLaren car rotating at a fixed rate is used as the scene. (b) Latency Breakdown shows that even without congestion traffic, the RTT takes up a large chunk of the total end-to-end latency and exhibits higher variance than other components. (c) Measured bitrate for different encoders.

IV. XR System Requirements

The requirements of the XR system for the measurement setup depicted in Fig. 2 were done using a high resolution car model, which the user wearing the HMD can experience from different angles and distances. A screenshot of the test scene is shown in Fig. 3a. The motion of the scene imitates the user behavior and ensures that the bitrate fluctuates over time, representing a real use case.

Fig. 3b depicts the individual time delays introduced by specific sections or components of the application. When summed up, these individual delays contribute to the total latency experienced by the user. These sections include WebRTC communication time, RTT over the network links, rendering, encoding, decoding, and wait times for decoding and rendering. The entry ”WebRTC” depicts the accumulated overhead caused by WebRTC on the server and the client. This is the time it takes from passing a finished frame to WebRTC for transmission to receiving the reassembled frame on the client device. It includes packet reordering, audio/video synchronization, and en/decryption and is assumed to be the fixed overhead of the underlying protocol.

The entries named ”Wait for ... ” are periods the application has to wait before processing the frame data to decode or display the frame. Generally, both entries should always be less than 1/FPS because a frame is displayed strictly in this interval. Therefore, a new frame should always be accepted for processing in this interval. In theory, the waiting periods could be reduced to about 0 ms when the timing between frame arrival, frame decoding, and displaying is tightened to always have a new frame ready at the display interval of the HMD.

The ”RTT” part of the latency is affected by other network traffic. Without any background load, the one-way delay is half of the RTT (3-5 ms). It already exhibits a significant portion of the total end-to-end latency and a higher variance than other sections. For many XR applications, a one-way latency budget of 10-200 ms at the network side is prescribed [7]. Therefore, the queuing latency in the network should be kept below this value even under congestion and competing flows. As shown in Section V, classical flows are unable to keep this latency budget under these bandwidth constraints.

Finally, Figure 3c shows different codecs’ utilized bandwidth at 60 FPS for this scene. H.264 exhibits a narrow range of bitrates around 40 Mb/s, while H.265 and AV1 have much lower and more consistent bitrates around 20 Mb/s and below. Thus, in Section V, we change the available bandwidth for the application around this value for classical, and L4S flows to ensure that the application will always demand more bitrate than the network can provide.

V. Results

A. Varying Network Bandwidth

This analysis focuses on the adaptability of L4S under fluctuating network conditions compared to a classical scenario. To assess this, the available network bandwidth is shaped between 5 and 20 Mbps. Fig. 4 illustrates the outcomes for both classical and L4S-enhanced flows, with the top figures representing classical flow behavior and the bottom depicting L4S capabilities.

The first column of Fig. 4 displays the bandwidth fluctuations alongside the encoder output. It highlights that while classical networking fails to adjust to these changing bandwidth conditions effectively, L4S-equipped flows dynamically respond to the current network state.

In the middle column, packet loss is quantified for each scenario. For the classical flows, three peaks can be observed, showing a packet loss of 0.4% attributed to excessive queuing

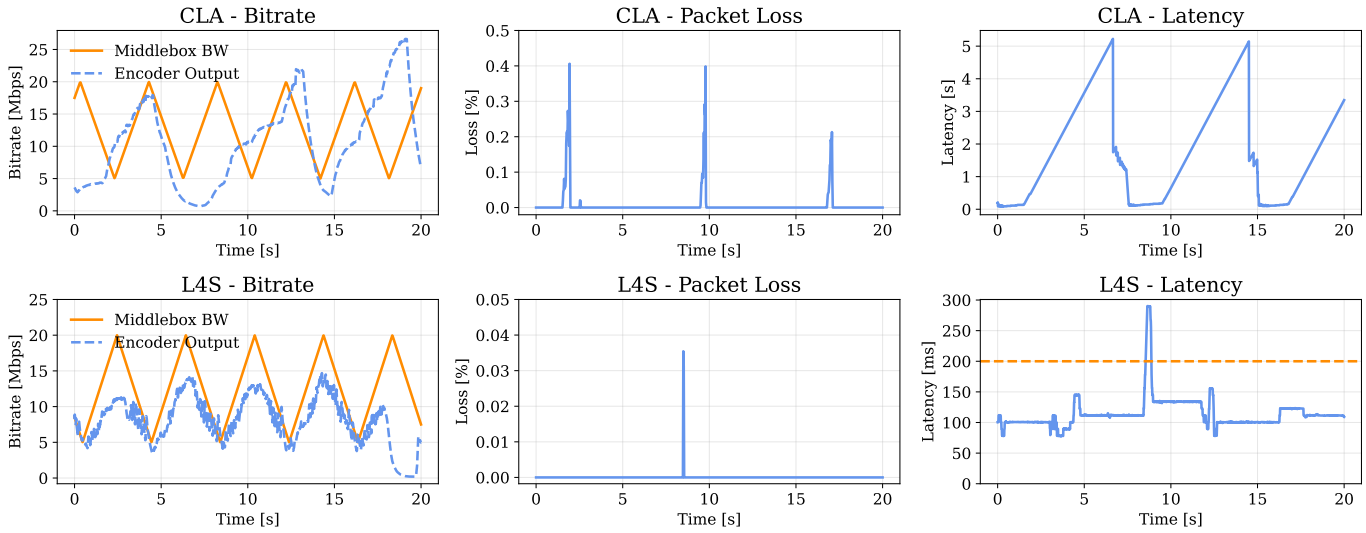

Fig. 4: Scenario with varying available bandwidth (between 5 and 20 Mb/s for the XR flow. In classical scenarios without L4S support, the latency increases to up to 5 s, resulting from queuing delays. With L4S, the latency stays low.

along the data path. In contrast, L4S flows demonstrate negligible packet loss due to its improved queuing management. The packet loss for a classical flow was over 10 times more than for the L4S flow. This differentiation is evident in the right column, where classical flows experience latency spikes up to 5 seconds due to queue saturation, while L4S maintains minimal latency and fluctuations, optimizing network performance under these conditions. The latency budget of 200ms [7] was violated just once when the flow also experienced packet loss.

B. Dynamic Response of L4S

Fig. 5 shows the queuing delay at the middlebox router of L4S packets under varying bandwidth conditions for the XR flow. Two levels of dynamicity were considered, one where the available bandwidth changed at a slower rate and another where it changed at a faster rate. The XR application with L4S and RTPrague has a reaction time of $\frac{1}{FPS} + \text{RTT}$ seconds. Under both levels, L4S flow managed to adjust its sending rate well. However, when the available bandwidth is throttled below 10 Mb/s, the queuing latency shoots up significantly for both levels of dynamicity. However, higher peaks are seen with more dynamic conditions. This latency spike is still within 200 ms but might not be suitable for highly interactive XR applications. The higher latency at lower bandwidth might be mitigated by fine-tuning the rate-control algorithm to respond more adaptively to rapid network changes.

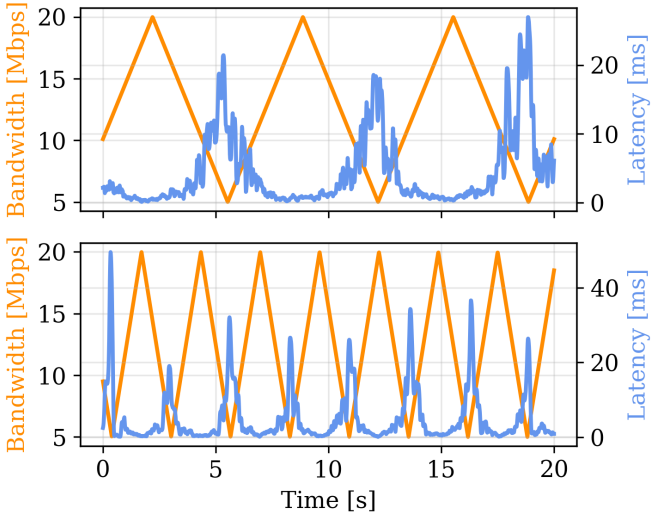

Fig. 5: Latency due to Queuing under dynamic changes in available bandwidth for L4S flows. The queuing latency for L4S packets shoots up when the available bandwidth is throttled below 10Mb/s. L4S can adjust to large and fast changes in available bandwidth.

C. Multi-User With Background Traffic

Figure 6 depicts latency and bitrate of the L4S flow where two devices are streaming the same scene simultaneously with and without background traffic. It can be seen that the overall latency is higher compared to Fig. 3(b) due to the extra load on the streaming server. However, the consumed bandwidth of a single stream does not change. The background traffic attempted to congest the link as much as possible. This has no noticeable effect on the latency or the bitrate of the L4S application, showing that when competing with classical flows, L4S flows can maintain their QoE even at full congestion from a competing classical flow.

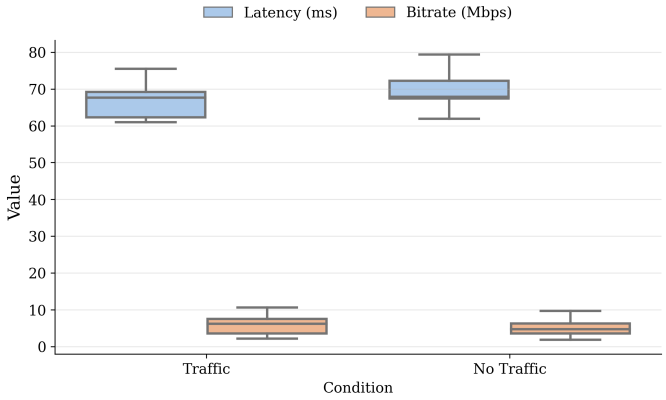

Fig. 6: L4S bandwidth and latency in a scenario with 2 XR devices with and without background load.

VI. Related Work

L4S architecture builds on the concept of Explicit Congestion Notification (ECN). Research on L4S covers the spectrum of both high throughput and ultra-low delay.

The L4S architecture is designed to support low-latency communication by reducing buffer bloat and providing congestion signaling. It builds on the concept of ECN (first introduced in RFC 2481 in 1999 [14]; reworked to its current state in RFC 8311 in 2018 [15]) marking and aims to offer scalability with minimal delay, suitable for modern applications such as gaming, real-time communications, and cloud services. Research on L4S often highlights the ability to deliver both high throughput and ultra-low delay, facilitated by DualQ Coupled AQM [16], which allows Classic and L4S traffic to fairly coexist. [17] have laid the groundwork for understanding L4S's efficacy, especially in environments where achieving consistently low latency is paramount.

On the other hand, classic AQM has been an important area of study in network performance. Early works on AQM, such as RED [18], were instrumental in introducing proactive congestion avoidance by managing queue lengths before packet loss occurs. Over time, various improvements have been proposed to address the shortcomings of RED, including PIE (Proportional Integral controller Enhanced) [19] and CoDel (Controlled Delay) [20], which aim to address issues like buffer bloat and maintain consistent queuing delays. [20]'s work on CoDel was particularly influential in redefining how networks handle queuing delays in environments dominated by TCP flows.

Recent research comparing L4S and CLA provides insights into the strengths and weaknesses of each approach. Some examples on that topic include the work by [21], titled 'Adaptable L4S Congestion Control for Cloud-Based Real-Time Streaming Over 5G' [21], which evaluates L4S performance in a cloud streaming context, demonstrating its adaptability and scalability in real-time applications. Additionally, [13] provides insights into L4S performance in hardware implementations. [22] also explored the viability of L4S in cellular environments and highlighted the challenges associated with such deployments.

Algorithms like DCTCP [23] and BBR [24] have shaped how low-latency and high-throughput requirements are managed. L4S is often positioned as a natural extension of DCTCP's mechanisms into broader internet environments, taking advantage of ECN-based congestion signaling. In contrast, traditional CLA approaches have often been associated with Reno or Cubic TCP, which rely on packet loss as a primary congestion signal.

A notable example that also enables streaming for XR devices is the work by [25], which explores the potential of L4S in enhancing XR/VR streaming performance over 5G networks. This study demonstrates how L4S can help to reduce latency in interactive XR applications, a crucial aspect for maintaining user immersion.

Finally, comparing deployment and practical use cases is essential to understanding the broader implications of these technologies. Works addressing real-world deployments, such as those by [26] and [27], have highlighted the practical benefits and challenges of adopting L4S and CLA AQMs at scale. Issues like interoperability, standardization, and the complexity of configuring network devices for optimal performance are key considerations frequently addressed in the literature.

To the best of our knowledge, this paper represents the first evaluation of L4S on a real XR device. Our findings provide novel insights into realistic evaluations for XR, accounting for its highly dynamic and variable traffic patterns.

VII. Conclusion

For a realistic evaluation of the latency, bandwidth, and packet loss performance of XR applications with remote rendering capabilities using L4S vs. classic networking, a testbed has been constructed. Our test setup shows that a bitrate of 50Mb/s is sufficient for a normal use case with state-of-the-art encoders. Particularly notable is the H.265 for which only a max bitrate of 20Mb/s was measured. A breakdown of the latency of each operation in the remote rendering stack shows that, under no network load, a maximum end-to-end latency of 50ms is achieved. However, very high delays of up to many seconds are observed for classical flows under bandwidth limitations. These high delays and packet losses cannot be sustained for a satisfactory QoE in most XR deployments. Our evaluation results show that L4S can lower queuing delays and effectively remove packet loss. In realistic testing environments, it successfully minimizes latency disruptions during dynamic network conditions and under congestion from background network traffic, thereby sustaining a smoother XR experience for users.

Acknowledgements

This work received funding from the Bavarian State Ministry for Economic Affairs, Regional Development and En-

ergy (StMWi) project KI.FABRIK (grant no. DIK0249) and the German Federal Ministry of Education and Research of Germany (BMBF) in the program of ”Souverän. Digital. Vernetzt.” joint project 6G-life, project identification number 16KISK002.

REFERENCES

[1] Y. Kim, Y. Choi, Y. C. Lee, H. Han, and S. Kang, “E-render: Enabling uhd-quality cloud gaming through edge rendering,” *IEEE Access*, vol. 10, pp. 72 107–72 119, 2022.

[2] A. Zoubarev, L. Bassbouss, D. L. Tran, S. Steglich, and S. Arbanowski, “A novel approach for remote rendering and streaming in xr,” in *2024 2nd International Conference on Intelligent Metaverse Technologies & Applications (iMETA)*, 2024, pp. 198–205.

[3] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve, “Illixr: Enabling end-to-end extended reality research,” in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021.

[4] J. Ratcliffe, F. Soave, N. Bryan-Kinns, L. Tokarchuk, and I. Farkhatdinov, “Extended reality (xr) remote research: a survey of drawbacks and opportunities,” in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3411764.3445170

[5] T. Theodoropoulos, A. Makris, A. Boudi, T. Taleb, U. Herzog, L. Rosa, L. Cordeiro, K. Tserpes, E. Spatafora, A. Romussi *et al.*, “Cloud-based xr services: A survey on relevant challenges and enabling technologies,” *Journal of Networking and Network Applications*, vol. 2, no. 1, pp. 1–22, 2022.

[6] B. Krogfoss, J. Duran, P. Perez, and J. Bouwen, “Quantifying the value of 5g and edge cloud on qoe for ar/vr,” in *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, 2020, pp. 1–4.

[7] 3rd Generation Partnership Project (3GPP), “Technical Report 26.929: EVS Codec Performance Characterization,” ETSI, Technical Report 26.929, July 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/126900_126999/126929/16.01.00_60/tr_126929v160100p.pdf

[8] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, “Low-latency networking: Where latency lurks and how to tame it,” *Proceedings of the IEEE*, vol. 107, no. 2, pp. 280–306, 2019.

[9] B. Briscoe, K. Schepper, M. Bagnulo, and G. White, “Low latency, low loss, scalable throughput (l4s) internet service: Architecture,” *RFC 9330*, 2023.

[10] K. Ramakrishnan, S. Floyd, and D. Black, “The addition of explicit congestion notification (ECN) to IP,” *RFC 3168*, 2001.

[11] K. D. SChepper, B. B., and W. G., “Dual-queue coupled active queue management (AQM) for low latency, low loss, and scalable throughput (L4S),” *RFC 9332*, 2023.

[12] B. Briscoe, “Prague congestion control,” Internet-Draft, July 2020. [Online]. Available: https://www.ietf.org/archive/id/draft-briscoe-iccrg-prague-congestion-control-03.html

[13] L. C. de Almeida, P. D. Maciel Jr, F. L. Verdi, and J. Pessoa-PB-Brazil, “Evaluating l4s framework performance with programmable data plane hardware.”

[14] D. K. K. Ramakrishnan and S. Floyd, “A Proposal to add Explicit Congestion Notification (ECN) to IP,” RFC 2481, Jan. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2481

[15] D. L. Black, “Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation,” RFC 8311, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8311

[16] K. De Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe, “Pi2: A linearized aqm for both classic and scalable tcp,” in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 105–119.

[17] B. Briscoe, K. D. Schepper, and M. Bagnulo, “Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture,” Internet Engineering Task Force, Internet-Draft draft-ietf-tsvwg-l4s-arch-00, May 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tsvwg-l4s-arch/00/

[18] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[19] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, “Pie: A lightweight control scheme to address the bufferbloat problem,” in *2013 IEEE 14th international conference on high performance switching and routing (HPSR)*. IEEE, 2013, pp. 148–155.

[20] K. Nichols and V. Jacobson, “Controlling queue delay,” *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[21] J. Son, Y. Sanchez, C. Hellge, and T. Schierl, “Adaptable l4s congestion control for cloud-based real-time streaming over 5 g,” *IEEE Open Journal of Signal Processing*, 2024.

[22] B. Mathieu and S. Tuffin, “Evaluating the l4s architecture in cellular networks with a programmable switch,” in *2021 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2021, pp. 1–6.

[23] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.

[24] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time,” *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[25] J. Son, Y. Sanchez, C. Hellge, and T. Schierl, “Split rendering with l4s over 5g for latency critical interactive xr applications,” *IEEE Communications Magazine*, vol. 62, no. 8, pp. 46–52, 2024.

[26] J. Holland, aug 2020. [Online]. Available: https://datatracker.ietf.org/meeting/interim-2020-maprg-01/materials/slides-interim-2020-maprg-01-sessa-latency-aqm-observations-on-the-internet-01

[27] B. Trammell, M. Kühlewind, D. Boppart, I. Learmonth, G. Fairhurst, and R. Scheffenegger, “Enabling internet-wide deployment of explicit congestion notification,” in *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings 16*. Springer, 2015, pp. 193–205.