

Scalability analysis and performance evaluation of WebRTC systems using the Infrastructure as Code paradigm and containerized traffic generators

Patryk Organiściak^{1*}, Mateusz Salach¹, Alicja Gerka¹,
Krzysztof Demidowski¹, Paweł Kuraś², Karolina Kisała³

¹ Department of Complex Systems, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, ul. M. C. Skłodowskiej 8, 35-036 Rzeszów, Poland

² Institute of Telecommunications and Cybersecurity, AGH University of Krakow, ul. Czarnowiejska 74, 30-059 Kraków, Poland

³ Faculty of Mathematics and Applied Physics, Rzeszow University of Technology, ul. Powstańców Warszawy 8, 35-029 Rzeszów, Poland

* Corresponding author's e-mail: org@prz.edu.pl

ABSTRACT

The dynamic development of remote education and distributed work has driven the demand for reliable real-time communication platforms. Ensuring the stability of such systems under massive load is a standard engineering challenge, necessitating robust methodologies for scalability assessment and proactive cloud infrastructure monitoring. Despite substantial progress in WebRTC benchmarking, practical gaps remain in the context of high-concurrency educational deployments. To achieve rigorous experimental repeatability and eliminate the configuration drift common in traditional load testing, the research adopted an advanced evaluation framework based on declarative infrastructure provisioning (Infrastructure as Code). The target environment for this evaluation was the BigBlueButton (BBB) system (version 3.0.19), deployed on a high-capacity server (e.g., 64 vCPU, 315 GiB RAM) as part of a major educational project in Poland. The traffic generation environment consisted of a distributed cluster of Docker containers, ensuring full isolation and repeatability of test scenarios. End-user simulation (utilizing headless Chromium clients and standard video resolution profiles) was implemented using scripts based on the Playwright library, allowing for the emulation of realistic interactions with the WebRTC interface. Iterative load tests at a scale of up to 421 users demonstrated the high efficiency of the server cluster, which provided empirical evidence that backend compute resources do not constitute the exclusive limiting factor in massive-scale deployments. The analysis revealed that the critical limitation of the system is the performance of the client-moderator's browser, which loses functionality beyond the threshold of 250 active video streams despite significant server resource redundancy. Quantitative metrics confirmed this imbalance: at peak load, server CPU utilization hovered between 12.1% and 16.2%, and RAM usage reached only 10 GiB out of 315 GiB, whereas external network traffic peaked at 374 Mb/s with a noticeable rise in TCP retransmissions and packet drops. The results confirm that the analyzed infrastructure is capable of stably supporting user groups that significantly exceed the initial design assumptions. The research suggests that for sessions exceeding 250 participants, engineering efforts should focus on client-side application optimization and local workstation resource management, rather than merely scaling server resources.

Keywords: load testing, real-time communication, Terraform, Docker, cloud resources, end-user simulation, automated testing methodology, distributed systems.

INTRODUCTION

The dynamic expansion of large-scale digital education has fundamentally increased the reliance on browser-based real-time communication (RTC) platforms. Ensuring the stability and reliability of these systems under massive, concurrent load (targeting groups of up to 421 concurrent users in this study) remains a significant engineering challenge, necessitating robust methodologies for scalability assessment [1] and proactive infrastructure monitoring [2]. Traditional functional testing methodologies often fail to capture the realistic performance bottlenecks that emerge under high-concurrency conditions. In complex WebRTC environments, scalability is governed by multiple interconnected layers, including signaling, media processing, and network transport. Consequently, standard assessments can be misleading; a backend server may appear technically operational with sufficient resource headroom, while client-side media rendering and interactivity drastically degrade.

Despite substantial progress in WebRTC benchmarking [2], critical practical gaps persist in the context of high-concurrency educational deployments. First, existing load testing campaigns frequently rely on manual execution or ad hoc scripts that lack proper infrastructure orchestration, which severely limits their reproducibility and methodological transparency. Furthermore, many studies isolate either protocol-level or platform-level metrics, failing to provide an integrated workflow that links infrastructure provisioning, synthetic user orchestration, and host-level observability within a single reproducible campaign [3]. Second, empirical evidence regarding the migration of bottlenecks between backend resources and client-side behavior under extreme concurrency remains sparse in applied educational scenarios. Finally, reproducible evaluation procedures that can be directly adopted into the daily operations of educational institutions remain fragmented across heterogeneous tools.

To bridge this gap, this study introduces an automated testing framework built upon the Infrastructure as Code (IaC) paradigm, which provides a unified workflow, contrasting with conventional approaches that rely on fragmented tools or manual execution. By establishing an integrated methodology based on declarative infrastructure orchestration and isolated runtime environments, realistic end-user media interactions are

effectively emulated at scale. BigBlueButton was selected as the target WebRTC conferencing stack due to its comprehensive API and representative media handling capabilities. Iterative load escalation was employed to pinpoint saturation thresholds, effectively differentiating server-side capacity limits from client-side interaction failures, with host-level metrics aggregated via Grafana.

Ultimately, this research provides a reproducible, engineering-oriented methodology tailored for WebRTC stress testing. The findings demonstrate that substantial backend resource headroom frequently coexists with severe transport stress and front-end interaction failures during extreme concurrency, offering actionable guidance for infrastructure operators managing synchronous digital teaching services.

While existing literature advances WebRTC benchmarking, it often bifurcates protocol and platform metrics without providing an integrated, IaC driven workflow for educational scenarios. The present study addresses these deficiencies through four contributions:

- An integrated, repeatable, IaC-based experimental framework for cloud-provisioned WebRTC stress testing, unifying automated infrastructure provisioning with host-level observability, which has been released as an open-source toolset for immediate adoption by the engineering community.
- A containerized, Playwright-based end-user emulation workflow for controlled load generation.
- An analysis distinguishing server-side headroom from client-side performance degradation under extreme load.
- Operationally actionable monitoring implications for continuous RTC governance.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 details the materials, methods, and the IaC evaluation framework. Section 4 presents the performance results. Section 5 discusses the findings and practical implications. Section 6 concludes the study.

RELATED WORK

Validating WebRTC applications at scale necessitates dedicated automation strategies, as traditional manual test execution is insufficient for complex distributed architectures [2,4]. Recent

experimental studies highlight the efficacy of black-box, browser-driven methodologies utilizing synthetic clients to obtain repeatable quality of service (QoS) and quality of experience (QoE) metrics in deployed RTC systems [3,5]. Advanced approaches to WebRTC benchmarking have also focused on QoE-driven service configuration [6], demonstrating that service continuity at the server level does not inherently guarantee an acceptable user experience during high-load scenarios [7].

Analytical experiments focusing on gateway and server behavior reveal that signaling and media orchestration components exhibit distinct performance profiles. For instance, comparative studies of SFUs (Selective Forwarding Units) [8] and the Janus WebRTC gateway [9,10] indicate that performance varies heavily based on architectural choices, such as deploying the server on Docker containers versus virtual machines [11]. Scalability-oriented field experiments further underscore that key performance indicators—such as throughput, delay, and overall conferencing quality—are intrinsically tied to the deployment context, including the underlying cloud topology [12–14] and prevailing network conditions “in the wild” [15]. Other studies have evaluated resource management [16] and specific mechanisms like receive-side real-time congestion control [17] to understand performance patterns. Comparative analyses between WebRTC and SIP [18], as well as experimental performance evaluations of WebRTC in mobile and cellular environments [19], provide further context for RTC optimization.

From the perspective of infrastructure engineering, executing reproducible, large-scale experimental tests increasingly requires declarative provisioning and automated deployment pipelines [20]. Experimental literature on Infrastructure as Code (IaC) testing emphasizes idempotence [21] and advocates for programmable infrastructure to enable rigorous repeatability. The deployment of automated performance testing within cloud architectures has been extensively investigated, with researchers proposing generator-based experimental approaches [22], autonomic load-testing frameworks [23], and specific architectures like svLoad [24].

Recent advancements highlight the necessity of experimental scalability assessments in microservice architectures and complex deployment configurations, leveraging operational profiles [25] and automated test-based learning [26]. Within the specific domain of browser-driven

end-to-end evaluations, modern tools that extend standard WebDriver architectures, such as Selenium-Jupiter [27], alongside specialized scalability testing scenarios for e-learning WebRTC platforms [28] have proven essential. Furthermore, improving the testing efficiency of Selenium-based load tests [29] and addressing the automation of Rich Internet Applications (RIA) in the cloud [30] remain active research areas. Parallel studies emphasizing experimental cost-effective load generation [31] and the deployment of secure, scalable real-time communications using Hybrid Information-Centric Networking (ICN) [32,33] conclude that a hybrid optimization approach is mandatory for production-ready operations. Advanced selection approaches based on composite QoS parameters [34] and rigorous performance evaluations across domains, including e-commerce platforms like WooCommerce [35], further validate the need for comprehensive scalability testing methodologies.

Recent studies continue to underscore the complexity of WebRTC performance evaluation under realistic load. For instance, the deployment of containerized WebRTC architectures in educational environments has demonstrated the growing necessity of isolating services to ensure scalability and prevent resource contention during massive concurrency evaluations [36]. Moreover, recent evaluations highlight the critical importance of scenario-based load testing for assessing true QoS and QoE at scale, as opposed to purely protocol-level simulations [37]. Additionally, emerging research on transport-aware QoE modeling emphasizes that network transport stress directly degrades client-side media rendering, frequently leading to playback freezes before server capacity is fully saturated [38].

MATERIALS AND METHODS

Declarative infrastructure orchestration for reproducible experiments

Cloud resources used for load generation were managed declaratively using Terraform. The provisioning workflow targeted Amazon Web Services and relied on Spot instance requests (deploying a total of 25 instances to generate the maximum load of 421 users) to enable cost-efficient scaling of synthetic participant capacity utilizing Ubuntu 24.04 LTS (x86_64) machine

images. Infrastructure definitions included instance launch parameters, security group rules, user-data bootstrap scripts, and container runtime initialization. The generated hosts were configured to retrieve media assets, start monitoring endpoints, and execute the load generator container with environment-specific parameters. The architecture of the solution is shown in Figure 1.

The experimental methodology utilizes declarative provisioning to orchestrate the client load generation layer, ensuring robust scalability evaluation. This declarative approach guarantees strict topological consistency across experimental iterations, ensuring that observed performance degradations are exclusively attributable to system stress rather than artifactual configuration variances.

Containerized traffic generators

To enforce strict isolation and deterministic execution of synthetic operational profiles, the load generation environment was encapsulated within standardized runtime containers based on Ubuntu 24.04 LTS (Noble). The image builds upon a Playwright-compatible base, packaging the automation script alongside pinned runtime dependencies. Utilizing this prebuilt container directly in deployment scripts enables rapid scaling across ephemeral cloud instances. The lifecycle of a single container is illustrated in Figure 2.

Container execution employed host networking where required to reduce unnecessary encapsulation overhead during high fan-out traffic emission. Runtime variables-controlled target URL, API credentials, bot count per host, and session duration. This model made scenario composition explicit and reproducible, enabling the same experiment to be rerun with controlled changes in concurrency parameters. Containerization also simplified fault isolation and post-run diagnostics by keeping logs and runtime behavior tied to a known image state.

End-user simulation

The emulation logic utilized robust UI selectors and API-generated authentication tokens to ensure deterministic, non-manual session orchestration. Session start was intentionally staggered to minimize synchronization artifacts and to reflect progressive arrival behavior rather than instantaneous join spikes.

Server architecture and target system

The system under test was BigBlueButton (version 3.0.19) running on Ubuntu 22.04.5 LTS, deployed as the conferencing backend receiving synthetic participant load. Session orchestration used official API endpoints for meeting creation

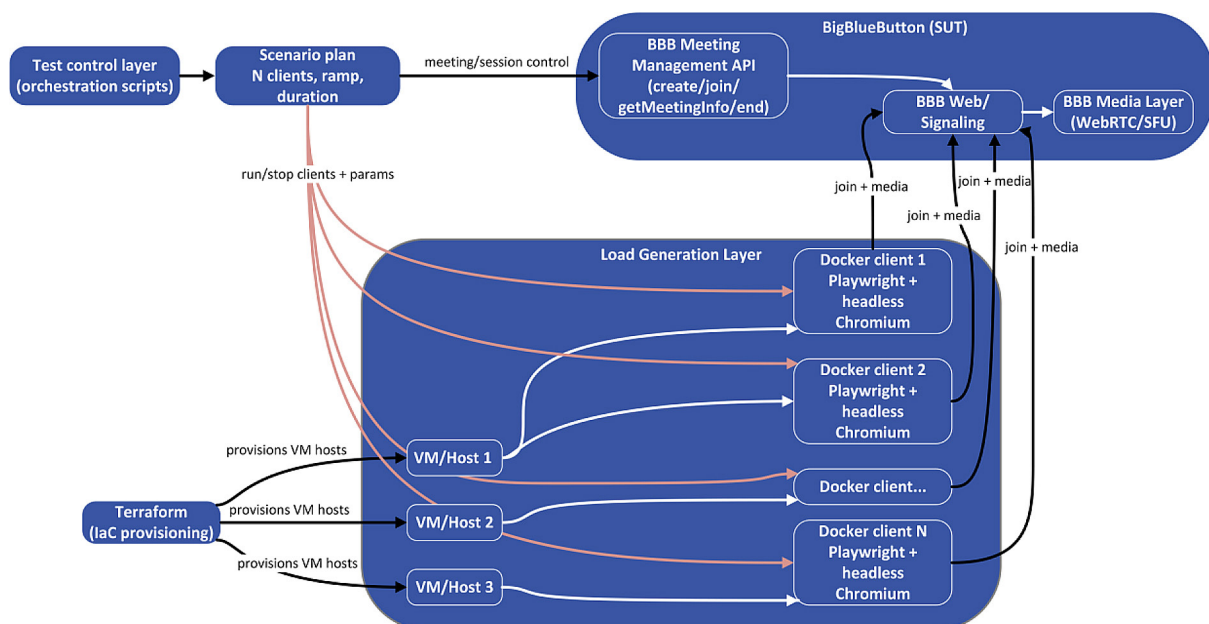


Figure 1. Architecture of the automated load testing framework, illustrating Infrastructure as Code (Terraform) resource provisioning, the distributed load generation layer (Docker, Playwright, headless Chromium), and the target communication system (BigBlueButton)

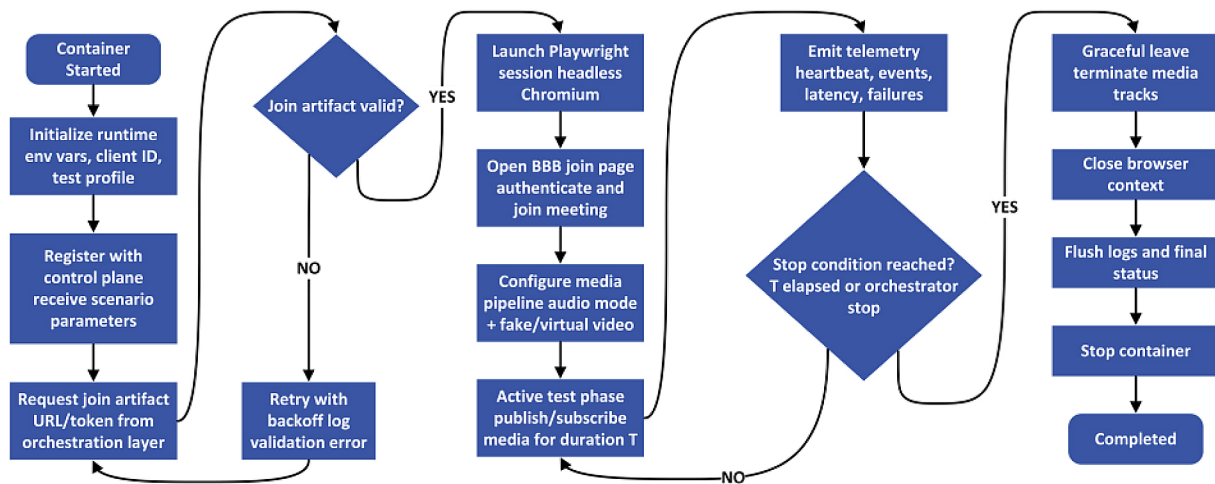


Figure 2. Life cycle of single client instance (Docker)

and participant joining, with checksum-based request signing in helper scripts. This ensured compatibility with platform-native control paths and eliminated ambiguity related to unsupported internal hooks. The presenter's perspective, essential for monitoring client-side degradation, was evaluated using a local LG gram workstation equipped with a 12th Gen Intel Core i7-1260P (2.10 GHz) processor, 16.0 GB of RAM, and Intel Iris Xe Graphics.

Observability data were collected via Grafana-linked host telemetry exports, covering CPU, memory, network throughput, transport error counters, queue indicators, and packet-drop series across escalating scenarios. This measurement strategy enabled comparison of service continuity against underlying transport stress and supported interpretation of whether resource pressure emerged on server compute/memory paths or in client/network behavior under high concurrency.

RESULTS

The load tests were conducted iteratively, scaling from 50 up to 421 concurrent users (Table 1). While server availability remained stable throughout the campaign, telemetry highlighted a clear imbalance in resource usage. Even under maximum load, CPU user-time utilization (Busy User) peaked at 12.1%, and memory usage barely reached 10 GiB out of the 315 GiB available. This confirms that backend compute and memory were nowhere near their limits.

Network transport metrics indicated significant stress, with external interface traffic peaking at 374 Mb/s. Escalating concurrency induced measurable network congestion, evidenced by rising TCP retransmissions, expanding UDP queues, and increased packet drop rates, despite stable server-side resource availability. These empirical observations reveal a distinct phenomenon of bottleneck migration under extreme concurrency. As backend

Table 1. Summary of performance metrics across load scenarios

Scenario	Users	CPU busy user (%)	RAM (GiB)	External Tx (Mb/s)	TCP retrans. (p/s)	Notes
1	50	2.7	7.7	33.3	-	Stage I baseline load
2	100	4.6	7.9	56.3	-	Stage II standard load
3	150	4.6	7.9	52.7	-	Stage III first latency anomalies
4	200	7.0	8.3	92.2	-	Stage IV high load
5	386	11.6	9.2	176.0	-	Stage V exceeding requirements (386 participants)
6	421	12.1	9.7	374.0	43.0	Stage VI max. network load (421 participants)

capacity remains robustly underutilized, the primary system constraints shift decisively towards front-end media rendering and client-side transport layer saturation. In addition to the performance metrics, a primary practical outcome of this research is the automated testing framework itself. The complete toolset (including Terraform configurations, Docker environments, and Playwright orchestration scripts) was consolidated into a Git repository. This approach ensures methodological transparency and enables other educational institutions or infrastructure operators to adapt the solution for their own scalability assessments (Figure 3).

DISCUSSION

The results confirm the tested server configuration had substantial resource headroom under large-scale load. Even at peak levels, CPU and memory utilization remained far below capacity, indicating backend resources did not limit practical scalability.

However, a critical client-side bottleneck emerged: browser performance and stability

degraded significantly beyond 250 concurrent connections from the presenter's perspective. This failure was detected scientifically through Playwright framework timeout exceptions during UI element interaction, accompanied by main thread unresponsiveness (screen freezing) observed directly on the workstation.

These findings indicate a bottleneck migration: as backend provisioning improves, constraints shift to front-end rendering, media-track handling, and browser overhead. Therefore, optimization must extend to client application efficiency, particularly UI recomposition and media-path tuning. The methodological originality of the IaC-based framework was meaningful to discovering this bottleneck migration. Benchmarking methodologies relying solely on browser-less traffic generators [31] typically evaluate only server-side capacity, which would have left the critical failure point, the moderator's browser undetected. Recent studies confirm that orchestrating browser-based performance evaluations under huge load is extremely resource-intensive, necessitating distributed infrastructures [39] By abstracting the infrastructure layer through Terraform



Figure 3. Server telemetry data at peak load, illustrating low CPU and memory consumption relative to total capacity, alongside a significant surge in network throughput on the external interface (above 400 client connections)

and containerizing synthetic Playwright clients, our approach provides a highly automated, cost-efficient solution to this challenge, enabling full-browser emulation at scale within a single, reproducible deployment pipeline.

Practical recommendations include: (1) prioritizing client-side UI optimization in high-density sessions; (2) incorporating generator vCPU and quota policy reviews into cloud scaling strategies to prevent orchestration constraints; and (3) monitoring transport and interaction indicators to detect early quality degradation. While the primary bottleneck in this study emerged on the client side, maximizing the resilience of the entire underlying architecture remains a fundamental engineering imperative. Overcoming endpoint limitations may require moving beyond traditional WebRTC topologies. For instance, the computational strain on the moderator's browser could potentially be alleviated by adopting edge computing task offloading strategies [39]. Furthermore, although server compute metrics remained stable, the observed surge in UDP queues highlights the necessity of advanced network observability. Integrating adaptive anomaly detection, similar to approaches in SDN [40], could provide dynamic traffic management, preventing transport-level congestion from exacerbating client-side degradation [41]. Such adaptive mechanisms are crucial for maintaining stability in critical infrastructure environments, where anomaly detection and automated system changes are required to ensure uninterrupted service delivery [42].

Omitting large-scale load tests prior to deployment represents a false economy. Although bypassing these tests might initially seem cost-effective, any functional failure during live educational sessions incurs substantial indirect costs. These include wasted time for presenters and large groups of participants, disrupted scheduling logistics, and sunk infrastructure rental expenses.

CONCLUSIONS

The IaC-driven architecture reliably generated high-volume WebRTC load during the performance evaluation. While the target Big-Blue-Button server infrastructure remained fully operational and significantly underutilized even above 400 concurrent connections, the practical scalability of the entire system was bounded by client-side constraints. Specifically, a critical performance bottleneck occurred at the moderator's

workstation, where browser stability and rendering capability drastically degraded beyond the threshold of 250 active video streams. Consequently, the empirical evidence demonstrated that backend resource headroom did not inherently guarantee end-to-end service scalability, as the limiting factors migrated directly from server-side compute paths to frontend transport and client-side media rendering processing, which aligned with phenomena observed.

For future research and practical implementations, it is highly recommended to shift optimization efforts toward mitigating endpoint resource exhaustion in high-density WebRTC sessions. Future studies should investigate edge computing paradigms and task offloading mechanisms to alleviate the decoding and UI rendering strain on presenter workstations. Additionally, implementing adaptive Software-Defined Networking (SDN) mechanisms for dynamic transport-level anomaly detection could proactively prevent UDP queue saturation and further improve the overall resilience of the end-to-end service delivery.

Acknowledgements

This research was supported by the European Union under the European Social Fund Plus, within the framework of the project "RaP STEAM - robotics and programming in primary schools of the Podkarpackie Voivodeship" (Fundusze Europejskie dla Podkarpacia 2021–2027). The authors would like to acknowledge the technical contribution of Power-Site Kamil Ślęzak, for providing technical support, specifically in the implementation of the load generation scripts and the execution of the automated test scenarios.

REFERENCES

1. Manishkumar Dave Amit D. Performance testing: Methodology for determining scalability of web systems. *International Journal of Science and Research (IJSR)* 2024;13:1254–61. <https://doi.org/10.21275/SR24121010827>
2. Vasar M, Srirama SN, Dumas M. Framework for monitoring and testing web application scalability on the cloud. *Proceedings of the WICSA/ECSCA 2012 Companion Volume*, New York, NY, USA: ACM; 2012, 53–60. <https://doi.org/10.1145/2361999.2362008>
3. Amirante A, Castaldi T, Miniero L, Romano SP. Jattack: a WebRTC load testing tool. 2016

- Principles, Systems and Applications of IP Telecommunications (IPTComm), IEEE; 2016, 1–6. <https://doi.org/10.1109/IPTComm39427.2016.7780247>
4. García B, Gortazar F, Lopez-Fernandez L, Gallego M, Paris M. WebRTC Testing: Challenges and practical solutions. *IEEE Communications Standards Magazine* 2017;1:36–42. <https://doi.org/10.1109/MCOMSTD.2017.1700005>
5. García B, Gortázar F, Gallego M, Jiménez E. User Impersonation as a Service in End-to-End Testing. *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, SCITEPRESS - Science and Technology Publications*; 2018, 707–14. <https://doi.org/10.5220/0006752207070714>.
6. Bertolino A, Calabro A, Angelis G De, Gortazar F, Lonetti F, Maes M, Tunon G. Quality-of-Experience driven configuration of WebRTC services through automated testing. *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, IEEE; 2020, 152–9. <https://doi.org/10.1109/QRS51102.2020.00031>
7. García B, Gortázar F, Gallego M, Hines A. Assessment of QoE for video and audio in WebRTC applications using full-reference models. *Electronics (Basel)* 2020;9:462. <https://doi.org/10.3390/electronics9030462>
8. Andre E, Le Breton N, Lemesle A, Roux L, Gouailard A. Comparative Study of WebRTC Open Source SFUs for Video Conferencing. *2018 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, IEEE; 2018, 1–8. <https://doi.org/10.1109/IPTCOMM.2018.8567642>
9. Amirante A, Castaldi T, Miniero L, Romano S Pietro. Performance analysis of the Janus WebRTC gateway. *Proceedings of the 1st Workshop on All-Web Real-Time Systems*, New York, NY, USA: ACM; 2015, 1–7. <https://doi.org/10.1145/2749215.2749223>
10. Park J, Jeong S, Kim SM, Kim J-S. Janus Streamer: A Scalable WebRTC Media Streaming Server. *2026 IEEE 23rd Consumer Communications & Networking Conference (CCNC)*, IEEE; 2026, p. 1–6. <https://doi.org/10.1109/CCNC65079.2026.11366482>
11. Spoiala CC, Calinciuc A, Turcu CO, Filote C. Performance comparison of a WebRTC server on Docker versus virtual machine. *2016 International Conference on Development and Application Systems (DAS)*, IEEE; 2016, 295–8. <https://doi.org/10.1109/DAAS.2016.7492590>
12. Xhagjika V, Escoda OD, Navarro L, Vlassov V. Load and Video Performance Patterns of a Cloud Based WebRTC Architecture. *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE; 2017, p. 739–44. <https://doi.org/10.1109/CCGRID.2017.118>
13. Jansen B, Goodwin T, Gupta V, Kuipers F, Zussman G. Performance evaluation of WebRTC-based video conferencing. *ACM SIGMETRICS Performance Evaluation Review* 2018;45:56–68. <https://doi.org/10.1145/3199524.3199534>
14. Chodorek RR, Chodorek A, Rzym G, Wajda K. A Comparison of QoS Parameters of WebRTC Video-conference with Conference Bridge Placed in Private and Public Cloud. *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, IEEE; 2017, p. 86–91. <https://doi.org/10.1109/WETICE.2017.59>
15. Varvello M, Chang H, Zaki Y. Performance characterization of videoconferencing in the wild. *Proceedings of the 22nd ACM Internet Measurement Conference*, New York, NY, USA: ACM; 2022, 261–73. <https://doi.org/10.1145/3517745.3561442>
16. Edan NM, Al-Sherbaz A, Turner S. Performance evaluation of resources management in WebRTC for a scalable communication. *Advances in Intelligent Systems and Computing*, 2019, p. 648–65. https://doi.org/10.1007/978-3-030-01177-2_48
17. Singh V, Abello Lozano A, Ott J. Performance Analysis of Receive-Side Real-Time Congestion Control for WebRTC. *2013 20th International Packet Video Workshop*, IEEE; 2013, 1–8. <https://doi.org/10.1109/PV.2013.6691454>
18. Parmar N, Ranga V. Performance analysis of WebRTC and SIP for video conferencing. *International Journal of Innovative Technology and Exploring Engineering* 2019;8:679–86. <https://doi.org/10.35940/ijitee.I1109.0789S19>
19. Moulay M, Mancuso V. Experimental performance evaluation of WebRTC video services over mobile networks. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE; 2018, 541–6. <https://doi.org/10.1109/INFCOMW.2018.8407020>
20. Bakhmetiev BS, Mykhalchuk HY. Infrastructure as a code approach for web infrastructure automation. *Actual Problems of Automation and Information Technology* 2024;28. <https://doi.org/10.15421/432401>
21. Hummer W, Rosenberg F, Oliveira F, Eilam T. Testing Idempotence for Infrastructure as Code, 2013, 368–88. https://doi.org/10.1007/978-3-642-45065-5_19
22. Jayasinghe D, Swint G, Malkowski S, Li J, Wang Q, Park J, Pu C. Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds. *2012 IEEE Fifth International Conference on Cloud Computing*, IEEE; 2012, 115–22. <https://doi.org/10.1109/CLOUD.2012.98>
23. Barna C, Litoiu M, Ghanbari H. Autonomic load-testing framework. *Proceedings of the 8th ACM international conference on Autonomic computing*, New York, NY, USA: ACM; 2011, 91–100. <https://doi.org/10.1145/1998582.1998598>

24. Noor J, Hossain MdG, Alam MA, Uddin A, Chellappan S, Al Islam ABMA. svLoad: An Automated Test-Driven Architecture for Load Testing in Cloud Systems. 2018 IEEE Global Communications Conference (GLOBECOM), IEEE; 2018, 1–7. <https://doi.org/10.1109/GLOCOM.2018.8647493>
25. Avritzer A, Ferme V, Janes A, Russo B, Hoorn A van, Schulz H, Menasché D, Rufino V. Scalability assessment of microservice architecture deployment configurations: A Domain-based approach leveraging operational profiles and load tests. *Journal of Systems and Software* 2020;165:110564. <https://doi.org/10.1016/j.jss.2020.110564>
26. Camilli M, Janes A, Russo B. Automated test-based learning and verification of performance models for microservices systems. *Journal of Systems and Software* 2022;187:111225. <https://doi.org/10.1016/j.jss.2022.111225>
27. García B, Delgado Kloos C, Alario-Hoyos C, Munoz-Organero M. Selenium-Jupiter: A JUnit 5 extension for Selenium WebDriver. *Journal of Systems and Software* 2022;189:111298. <https://doi.org/10.1016/j.jss.2022.111298>
28. Seyed C, Bilong JRN, Ouya S, Nanne MF, Niang I. Scalability and Performance Testing of an E-Learning Platform Integrating the WebRTC Technology: Scenario “Authentication.” *Advances in Intelligent Systems and Computing*, 2020, 186–93. https://doi.org/10.1007/978-3-030-11932-4_19
29. Shariff SM, Li H, Bezemer C-P, Hassan AE, Nguyen THD, Flora P. Improving the Testing Efficiency of Selenium-Based Load Tests. 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), IEEE; 2019, 14–20. <https://doi.org/10.1109/AST.2019.00008>
30. Snellman N, Ashraf A, Porres I. Towards Automatic Performance and Scalability Testing of Rich Internet Applications in the Cloud. 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE; 2011, p. 161–9. <https://doi.org/10.1109/SEAA.2011.33>
31. Gortázar F, Gallego M, Maes-Bermejo M, Chicano-Capelo I, Santos C. Cost-effective load testing of WebRTC applications. *Journal of Systems and Software* 2022;193:111439. <https://doi.org/10.1016/j.jss.2022.111439>
32. Papalini M, Carofiglio G, Compagno A, Mantellini A, Muscariello L, Samain J, Sardara M. On the Scalability of WebRTC with Information-Centric Networking. 2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), IEEE; 2020, 1–6. <https://doi.org/10.1109/LANMAN49260.2020.9153228>
33. Muscariello L, Papalini M, Roques O, Sardara M, Tran Van A. Securing scalable real-time multiparty communications with hybrid information-centric networking. *ACM Trans Internet Technol* 2023;23:1–20. <https://doi.org/10.1145/3593585>
34. Kheradmand M, Motameni H. An approach to web service selection based on composite QoS parameters. *Advances in Science and Technology Research Journal* 2015;9:129–36. <https://doi.org/10.12913/22998624/2379>
35. Organiściak P. WooCommerce in ecommerce - overall system review and stress tests. *Zeszyty Naukowe Wyższej Szkoły Technicznej w Katowicach* 2023;17:143–56. <https://doi.org/10.54264/0080>
36. Kamis A, Zakic A, Popovic G, Bogavac M, Ignjatovic B, Lakhmi J. Development of a cloud-based WebRTC VoIP application using the docker platform for an educational environment. *International Journal of Cognitive Research in Science, Engineering and Education (IJCRSEE)* 2026;14:11–21. <https://doi.org/10.23947/2334-8496-2026-14-1-011-021>
37. Huda AT, Purba DER, Singarimbun RN, Chaizir M, Apriandy KI. Analysis of WebRTC video streaming scalability in a broadcaster-viewer model using scenario-based load testing. *JUKI : Jurnal Komputer Dan Informatika* 2025;8:269–76. <https://doi.org/10.53842/juki.v8i1.2607>
38. Gurses F, Sedal A, Soyak EG. Transport-aware QoE modeling for WebRTC: Predicting playback freezes with machine learning. *Journal of Network and Systems Management* 2026;34:97. <https://doi.org/10.1007/s10922-026-10076-z>
39. Palaniappan SC, Ponnuswamy PP. Task offloading in edge computing using integrated particle swarm optimization and genetic algorithm. *Advances in Science and Technology Research Journal* 2024;19:371–80. <https://doi.org/10.12913/22998624/195658>
40. Organiściak P, Kuraś P, Strzalka D, Paszkiewicz A, Bolanowski M, Kowal B, Ćmil M, Dymora P, Mazurek M, Vanivska V. Detection of incidents and anomalies in software-defined network – Based implementations of critical infrastructure resulting in adaptive system changes. *Advances in Science and Technology Research Journal* 2024;18:176–91. <https://doi.org/10.12913/22998624/192641>
41. Bolanowski M, Paszkiewicz A, Żabiński T, Piecuch G, Salach M, Tomecki K. System architecture for diagnostics and supervision of industrial equipment and processes in an IoE device environment. *Electronics (Basel)* 2023;12:4935. <https://doi.org/10.3390/electronics12244935>
42. Klasa T, Fray I El. Load-balanced Integrated Information Security Monitoring System, *Annals of Computer Science and Information Systems*, 2017, 213–21. <https://doi.org/10.15439/2017F463>