Evaluating Browser-Based Networking for Real-Time Multiplayer Games

Daniel Orlando*
Colgate University

Aaron Gember-Jacobson
Colgate University

1 Introduction

Web browsers' support for full 3D graphics (e.g., WebGL and WebGPU) make browsers a viable platform for gaming. However, development real-time multiplayer games for browsers is challenging due to browsers' limited networking options. In this work, we evaluate the suitability of three browser networking options–WebSockets, WebRTC, and WebTransport–for competitive real-time multiplayer games, comparing their performance under typical game networking workloads. This differs from prior works on networking for games [3,5,6] which focus on non-browser-based games.

2 Networking requirements

Real-time multiplayer games (e.g., Fortnite, Valorant) often rely on *tick-based simulations* that advance the game state in discrete, fixed time intervals (e.g., 30 ticks per second) [1]. On each tick, a client transmits its inputs (e.g., player movement) to a central server. To account for the one-way delay (OWD) from the clients to the server, the server's simulation lags behind the clients' simulation. Under ideal conditions, all clients' inputs for tick n arrive at the server just before the server simulates tick n. After processing clients' inputs for tick n, the server sends the authoritative game state (e.g., positions of all players) back to clients. Clients process these updates at their next tick—which may be $> n+1$ depending on the round-trip time (RTT) relative to the gap between ticks.[1] Figure 1 depicts this sequence of events.

To achieve a better quality of experience (QoE), the networking layer must provide: (1) *low latency* to permit higher tick rates and minimize the lag between the clients' simulation of tick n and the clients' receipt of the server's authoritative game state for tick n, thereby reducing potential discrepancies

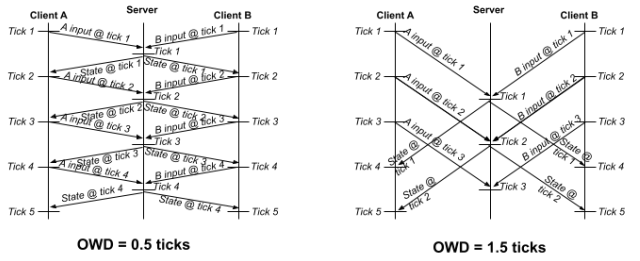

Figure 1: Example tick-based simulations

that must be reconciled; (2) *low jitter* to minimize the likelihood of clients' inputs for tick n arriving substantially before or after the server's simulation for tick n, thereby reducing the queuing or discarding, respectively, of inputs; and (3) *low loss* to minimize the need for the server to extrapolate clients' inputs, thereby reducing potential discrepancies between the clients' simulation and the server's simulation.

3 Browser-based networking

While non-browser-based real-time games typically use UDP secured with DTLS, web browsers offer three networking options—WebSockets, WebRTC, and WebTransport—each with unique traits that impact their ability to meet the aforementioned requirements.

WebSockets [4] are widely supported and enable flexible communication over a TCP connection. TCP's reliable delivery eliminates loss, but retransmitted updates incur higher latency and may arrive after the server has processed the relevant tick, causing the server to ignore the updates and the retransmission to be fruitless. Additionally, the congestion control algorithms commonly used by TCP are optimized for throughput instead of latency.

WebRTC [2] is designed for real-time, peer-to-peer media delivery. WebRTC offers UDP-like semantics to avoid fruitless retransmissions and employs a congestion control algorithm optimized for latency. However, WebRTC involves many layers of protocols, which increases data processing overhead.

*Undergraduate student author

[1] If the RTT is greater than the time between ticks, clients typically extrapolate the game state for tick $n+1$, $n+2$, ... $n+k$ based on the most recent update from the server, and reconcile any differences when the server's update for tick n is processed by the client k ticks later.

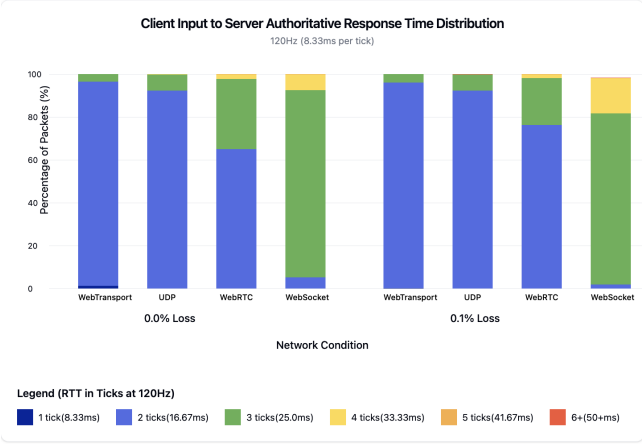

Figure 2: RTT distribution for all tested protocols under different loss conditions.

WebTransport [7] is an emerging standard that leverages QUIC's modern networking capabilities to support UDP-like datagrams. WebTransport's support for for unreliable communication eliminates the latency and jitter impacts of retransmissions, and WebTransport's congestion control (BBRv1) is optimized to low-latency.

4 Comparison of protocol performance

To assess each protocol's performance, we ran a simple tick based simulation in Rust using open source transport libraries for WebSockets, WebRTC, and WebTransport. As a baseline, we also tested a plain UDP socket secured with DTLS, to reflect the kind of networking we might see in a standalone multiplayer application outside of the browser constraints.

Setup: We use a research server at our university in upstate New York and a DigitalOcean server in New York city to simulate real network conditions. We conducted 3-minute tests in a network simulation running at 120 ticks per second, under ideal (0.0% packet loss) and degraded (0.1% packet loss) network conditions. Packet loss was implemented using an eBPF XDP program that randomly dropped packets at the specified loss rate. We quantify performance in terms of the number of ticks elapsed between the tick at which the client sent each input and the tick at which the response from the server was processed. A lower number of elapsed ticks translates to a lower feasible lag between the simulations on the client and server, leading to a better QoE.

Results: Figure 2 summarizes the measured latencies under different conditions. WebTransport outperforms all other protocols in both loss scenarios. Its performance advantage over raw UDP likely stems from its BBRv1 congestion control implementation. Despite utilizing UDP data channels, WebRTC's complex protocol stack hinders its speed and performance. WebSockets has the highest latency across both optimal and degraded network conditions. In the packet loss scenario, Figure 2 clearly demonstrates the negative impact of TCP's retransmission mechanisms on real-time multiplayer games.

5 Future work

Our future work will focus on a few areas. First, we plan to extend our protocol evaluations from controlled server environments to actual web browsers to assess performance on an end-user's device. Second, our current experimental setup uses a stable, high-capacity connection, which doesn't reflect typical user experiences. Testing each protocol across simulated residential connections or mobile networks (e.g, 3G, 4G, LTE) and multiple geographic vantage points would provide valuable insights into their performance under variable network conditions. Finally, we plan to investigate how specific mechanisms in each transport's congestion control algorithms are impacting their performance for real-time gaming.

References

[1] DEWET, M., AND STRAILY, D. Peeking into valorant's netcode. https://technology.riotgames.com/news/peeking-valorants-netcode. July 28, 2020.

[2] JESUP, R., LORETO, S., AND TÜXEN, M. WebRTC Data Channels. RFC 8831, Internet Engineering Task Force, Jan. 2021.

[3] KÄMÄRÄINEN, T., AND PIHLAJAMÄKI, M. Aspects of networking in multiplayer computer games. *International Journal of Computer Games Technology* (2017). Accessed: 2024-12-13.

[4] MELNIKOV, A., AND FETTE, I. The WebSocket Protocol. Tech. Rep. 6455, Internet Engineering Task Force, Dec. 2011.

[5] R., S. S., AND S., A. D. Analyzing the network traffic requirements of multiplayer online games. *IEEE Transactions on Network and Service Management* (2008). Accessed: 2024-12-13.

[6] SAIEDIAN, S., AND HASHMI, S. An evaluation of videogame network architecture performance and security. *Computer Networks* (2021). Accessed: 2024-12-13.

[7] VASILIEV, V. The WebTransport Protocol Framework. Internet-Draft draft-ietf-webtrans-overview-09, Internet Engineering Task Force, Feb. 2025. Work in Progress.