Evaluating Browser-Based Networking for Real-Time Multiplayer Games

Daniel Orlando and Aaron Gember-Jacobson

1. Motivation

Why browser gaming?

- 🌐 **Massive reach**: Browser games offer instant play and no installation
- 🕹️ **Challenge**: Competitive multiplayer games are considered unplayable above 100ms ping
- ❓ **Unknown:** Can browser technologies achieve the consistent low latency required for competitive gaming?

2. Multiplayer Background

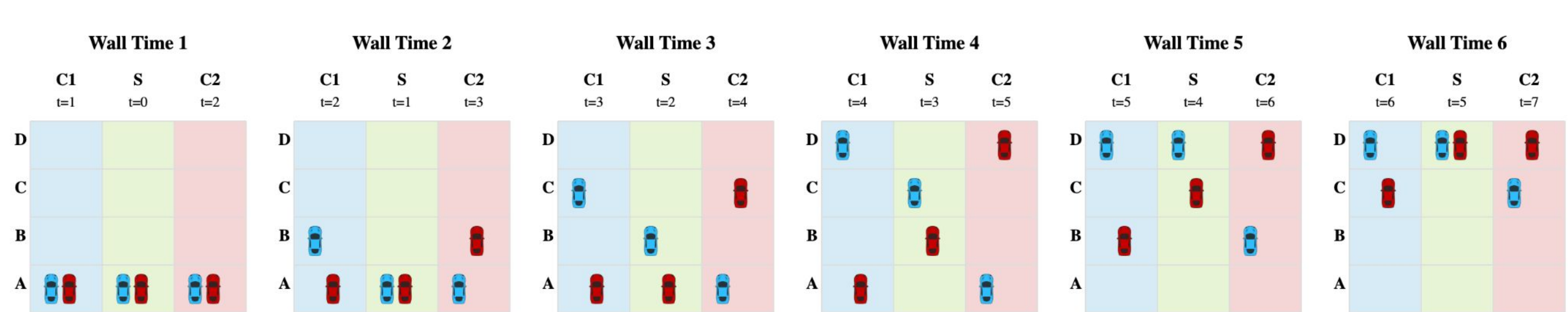
⏱️ **Tick-Based Simulation:** Multiplayer games discretize time into fixed intervals ("ticks") to maintain deterministic state synchronization across distributed clients.
🏇 **Clients run ahead of Server:** Client's run one-way-delay (OWD) in ticks ahead of server so that input for tick N arrives just in time for server processing tick N.

Network Condition Effects

🏎️ **Racecar Example:** Consider a game with two clients, each controlling their own race car. The first to get to point D wins.

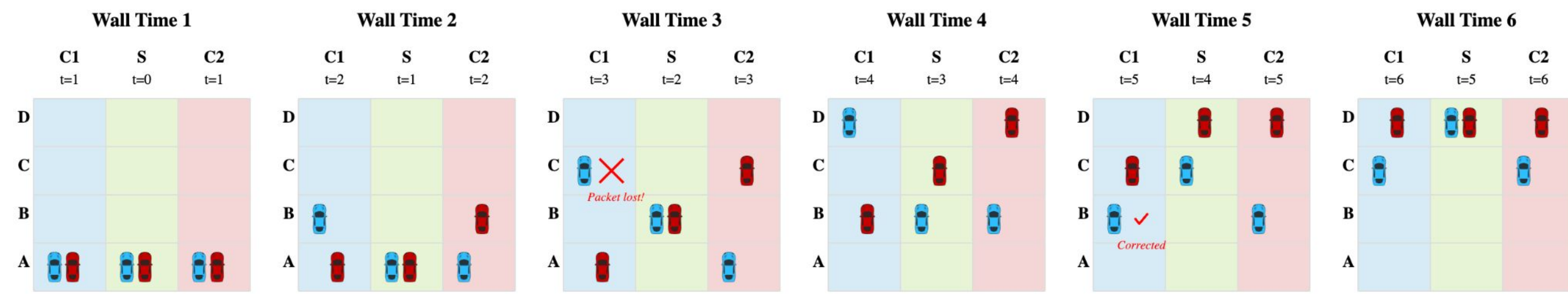
Latency Effects

Setup: Two clients with different network delays (C1: 1 tick OWD, C2: 2 ticks OWD)
Result: Lower-latency client always wins



Loss Effects

Setup: Two clients with same OWD, but Client 1 experiences loss
Result: C1 would visually stutter and C2 would win



3. Web Realtime Protocols

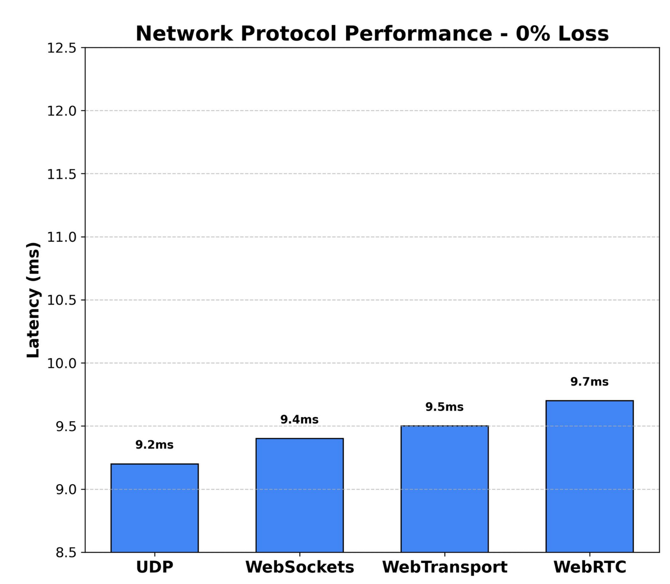
Tick-based simulations need low-latency, bi-directional communication. We evaluate three browser-native protocols.

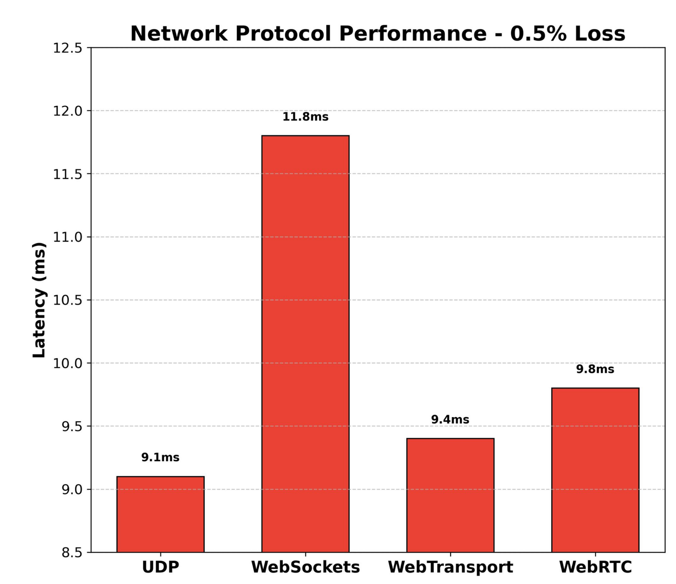
Feature	UDP w/ DTLS	WebSockets	WebRTC	WebTransport
Stack	DTLS	TCP	SRTP, RTP, DTLS	QUIC
Datagram Support	✓	✗	✓	✓
Browser Support	✗	✓	✓	-
Congestion Control	None	Cubic	GCC	BBR

4. Methodology

Tick Based Setup:

🧑‍💻 **Client**: Colgate (Hamilton NY)
🌐 **Servers**: NYC (low latency), SF (high latency)
🧪 **Protocols**: WebSockets, WebRTC, WebTransport, UDP with DTLS (as baseline)
⏱️ **Measurement**: Record response times per tick for each transport across ten 3-minute sessions over a 12-hour period.



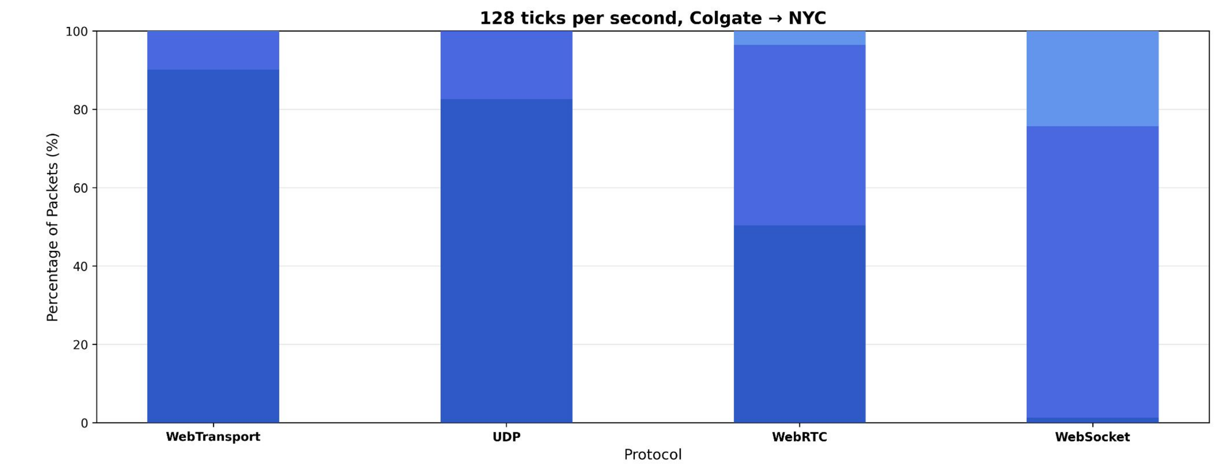


Naïve Setup - blast packets back and forth

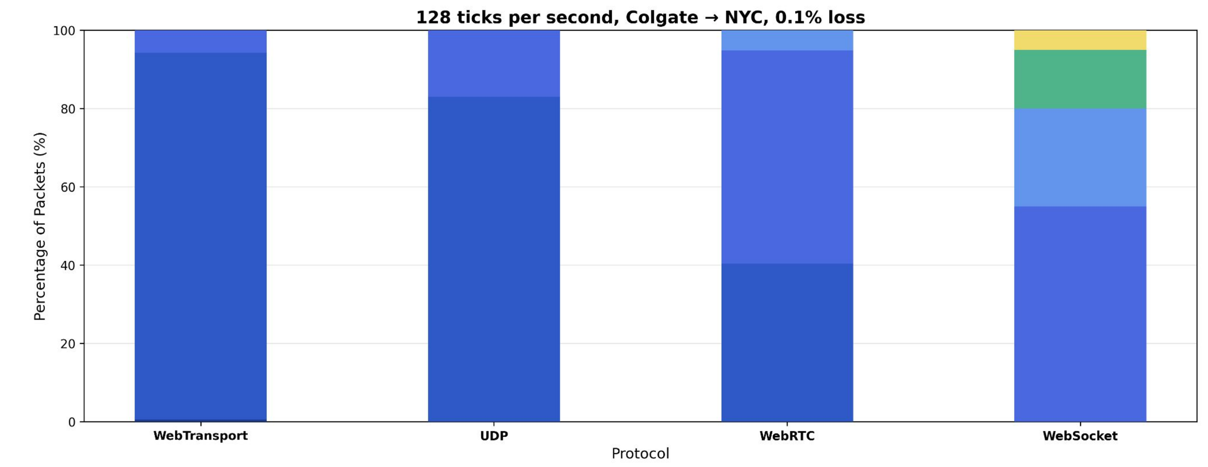
With no tick based simulation, initial benchmarks would tell us that all three protocols are quite similar

5. Results

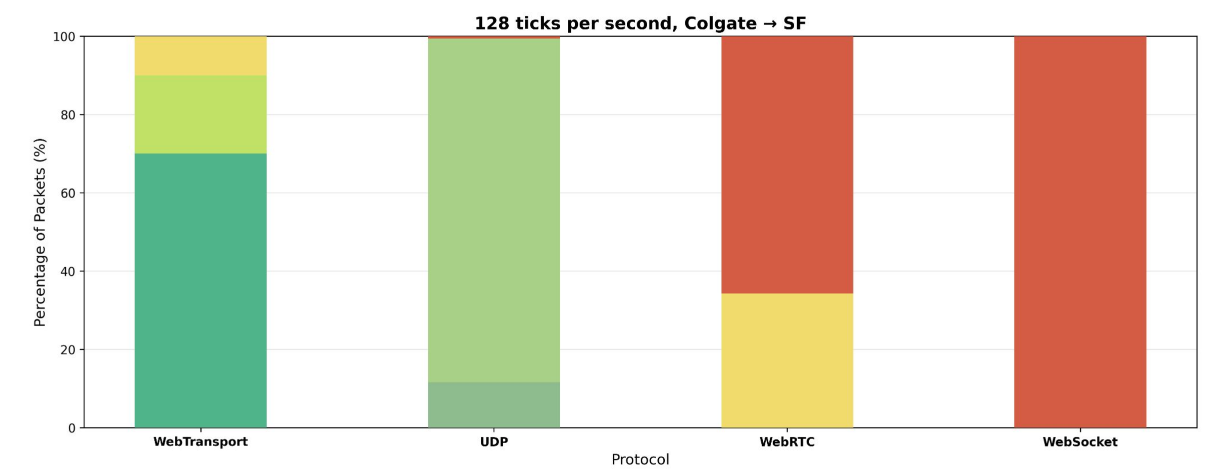
Baseline Interpretation: Under ideal network conditions, all 4 protocols perform nicely



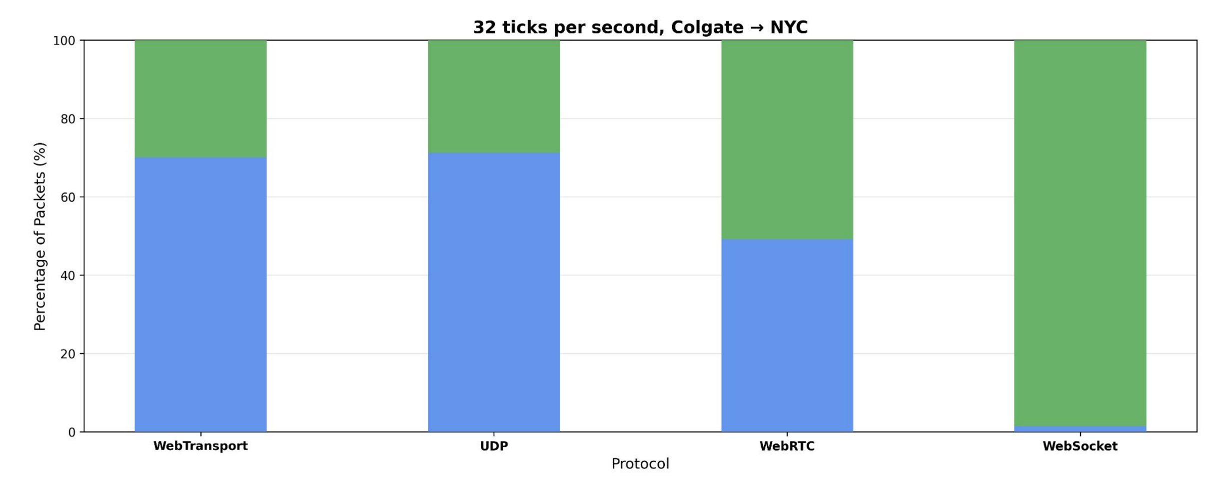
Loss Interpretation: As soon as loss is introduced, TCP WebSockets begin to suffer

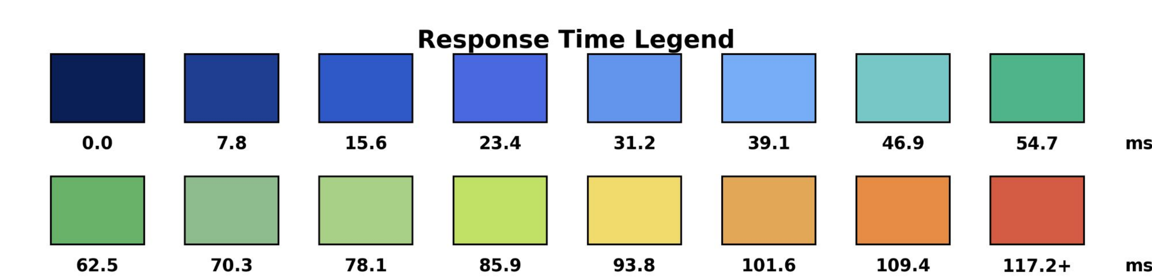


Latency Interpretation: WebTransport leads all web protocols but still trails UDP.



Tick-vary Interpretation: Lower tickrates stretch time between ticks which increases per-tick latency.





Key Takeaways:

✅ **WebTransport**: Consistently best performance under varying conditions.
❌ **WebSockets**: Poor for real-time multiplayer due to reliability & TCP.
❌ **WebRTC**: High protocol overhead, doesn't perform well in loss scenarios because of CC.