Distributed gaming arch considerations: low latency in regards to large physical distance (responsiveness), coherent game states (rollback, centralized authority),

Synchronous (real-time) vs async (turn based) with above considerations

https://en.wikipedia.org/wiki/Correspondence_chess

P2p fighting game network with rollback algo
https://youtu.be/7jb0FOcImdg
- https://www.ggpo.net/
- https://www.youtube.com/watch?v=0NLe4IpdS1w

Counter strike tick server (client-server):
https://www.youtube.com/watch?v=aWtPpry5UPg
https://cs2pulse.com/tick-rate-subtick-system-explained/
https://youtu.be/GX4595KeZyc

Authoritative servers:
- Dedicated servers that have the final say
- Examples: League of Legends, Apex Legends, Fortnite, etc.

Listen servers:
https://www.reddit.com/r/unrealengine/comments/1fmgd1u/peer_to_peer_vs_listen_server/
https://docs-multiplayer.unity3d.com/netcode/current/learn/listen-server-host-architecture/

Considerations for distributed gaming architecture

- large game assets transfer and integrity (game downloads, patching, file alteration check)
    - Popular multiplayer games range from several hundred megabytes to several hundred gigabyte (call of duty black ops 6 ~200GB, final fantasy 14 ~140GB)
    - Patches for new content may also be several hundred GB
    - Rolling daily player count range from thousands to millions; total size to transfer
    - World of warcraft uses a bittorrent like p2p protocol for downloading assets alongside
- user data (legal requirements like gdpr, personal information security, user authentication for login and payment)
- player interaction (low latency/good responsiveness, consistent (enough) and synchronized game state, server load distribution, input/action processing, inter-server communication, appropriateness for game mechanics (realtime vs turn based async)
- private

Richard Hu, Mingwei Zhang, Nathaniel Tang
Fluture
CS344/715
2025 July 28

Distributed Gaming Systems

Organizations that manage multiplayer games have 3 core considerations

1. **User account management and security (similar to non-gaming services)**
- Sensitive user data have to be protected and conformed to laws of the regions games wish to service: GDPR (EU: pop 450m), Ministry of Industry and Information (PRC: pop 1.4b), COPPA (US: pop 340m), etc.
- Financial transaction data protection requirements from payment vendors like Paypal, Visa, Mastercard, etc and legal like fraud detection, money laundering, etc.
- User account management systems will follow typical transaction database requirements like correctness, consistency, scalability, resilience

2. **Content distribution (similar to large media services)**
- Current popular games are larger than ever and will continue to grow in disk size; Fortnite (Epic Games, ~140GB), Call of Duty Black Ops 6 (Activision, ~240GB multiplayer), Counter-Strike 2 (Valve, ~60GB), Final Fantasy 14 (Square Enix, 140GB min)
- Tens to hundreds of thousands of active players at any given time each needing local installs
- New content, update, bugfix patches may eclipse the original install size; data integrity through error checking and using TCP is paramount
- Digital game distribution platforms (Steam, Epic Games, Good Old Games, Activision Blizzard, Xbox, Playstation, Nintendo, etc.) also manage millions of other games that will be downloaded by millions of global machines.
- CDNs will use a mix of P2P networks (BitTorrent-esque), database sharding and caching across datacenters (external and internal) and ISPs, to saturate client download speeds and propagate content. Can also be done at hyperlocal scale if last mile download is a bottleneck (home LAN parties, gaming events at convention centers and stadiums, LAN cafes) through routing/server software like LANCache/SteamCache which redirect user download DNS requests to a local cached copy on LAN.

3. **Multi-client networking for gameplay (focus of presentation and differs greatly from other distributed systems use cases)**
- System architecture is heavily dependent on gameplay; 1:1 fighting games (P2P), 1:N FPS/MOBA/MMOs (client-server server-authoritative)
- Round trip time (RTT) latency management may be extremely important for local game feel (real time FPS) or less important (turn-based Civilization, online chess); trade off with multi-client consistency

- Additional non-gameplay/non-functional requirements like cheating detection, player DoS mitigation, QoS, may drive choice of one system arch over another

Case 1: Counter Strike Global Offensive (CSGO): developed and published by Valve Corp.; reworked and renamed to CS2 in 2023.

Colohan, C. (2018). L10: How Counterstrike Works (Time in Distributed Systems) [Video L10: How Counterstrike Works (Time in Distributed Systems)]. distributedsystemscourse.com. https://www.youtube.com/watch?v=GX4595KeZyc
CSGO was reworked and renamed to CS2 in 2023.

Bharambe, A., Douceur, J. R., Lorch, J. R., Moscibroda, T., Pang, J., Seshan, S., & Zhuang, X. (2008). Donnybrook. ACM SIGCOMM Computer Communication Review, 38(4), 389–400. https://doi.org/10.1145/1402946.1403002

- CSGO is 5v5 objective shooter, each player sends to central server inputs, server responds with a resulting game state from calculating those inputs on a prior game state.

- with only client-server, players could only respond to what is sent by server (server-authoritative) and only reflects the past, not the present; if RTT is 200ms, equivalent to playing at unpleasant 5 fps (film 24, common online video 30, minimum acceptable playability for online competitive games 60)

- also have to deal with players receiving server game state (server tick) and sending inputs at different times (if server is in NJ, NY player will have lower latency than Australia player), out of order packet receipt also needs to be dealt with (implemented with logical clock sync/Lamport clock between players and servers)

- in between ticks, developers implemented predictive execution on the received game state (speculative execution; analogous to branch prediction in pipelined processors) to guess the next state calculated by the server. If next received game state is close enough to guesses, already calculated and cached, but if not, throw away incorrect guesses and run calculations starting from the received state. Predictive exec. only possible because finite inputs with finite game states during any given tick. Human behavior is also predicted: if currently a player is moving, likely will keep moving; if not moving, then likely to stay not moving. Goal is to make corrections on deviation from correct state nearly unnoticeable to players. Implemented by having each player run a local copy of the server state to run the predictions. Turns client-server from strongly consistent to weakly consistent to improve performance.

- gameplay mechanics like random weapon spread and cast-time before action exec (League of Legends) serve as latency perception compensation tricks. Server sending over only relevant game state info (send a player only nearby environment and other players game state, not those far away, unseen, or unchanged) reduces packet size.

- P2P systems like Colyseus and Donnybrook for similar multiplayer systems have been researched to remove the latency from a server hop, but increase network traffic O(n^2) and make cheat detection harder to implement (also publishers want centralized control). (Bharambe)

Case 2: Mortal Kombat X/Injustice 2: dev NetherRealm Studios, pub Warner Bros. Games

Stallone, M. (2019). 8 Frames in 16ms: Rollback Networking in Mortal Kombat and Injustice 2 [Video *8 Frames in 16ms: Rollback Networking in Mortal Kombat and Injustice 2*]. Game Developers Conference (Ed.), *Game Developers Conference.* https://www.youtube.com/watch?v=7jb0FOcImdg
Stallone, M. of NetherRealm Studios; presented at GDC 2018 (2018 Mar 19-23)
Cannon, T. (2022, September 25). pond3r/ggpo. GitHub. https://github.com/pond3r/ggpo

- Mortal Kombat X (2015) and Injustice 2 (2017) are 1v1 fighting games. P2P system where both peers agree on game state. Peers only send inputs to each other after initial game state sync. Initial matchmaking is client-server.

- Both are hard 60hz games; game state calculation and sending of information have to be done within hard 16ms window.

- Matches cannot exceed a defined amount of latency between peers (333ms) otherwise the engine forces a disconnect.

- Only waiting for the inputs from remote opponent before being able to respond is unacceptable by players (lockstep). Devs implemented local-side prediction much like CSGO's predictive exec in between input packets, however a last agreed upon game state (confirmed frame) is calculated from a prior confirmed frame and remote inputs. That most recent confirmed frame is deterministic (finite inputs possible between confirmed frames). On prediction miss, go back to the most recent confirmed frame and replay from remote inputs (rollback algo). Local-side optimizations allow more predictions to be calculated within 16ms.

- Desyncing and out of order packets between peers are risks; slowing down the further ahead peer to give behind peer time to resync. Modification of logical clock sync adjustment.

- Other rollback algs are used in other fighting games; GGPO's source code has been open source since 2019.