Computer Networks 192 (2021) 108128

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

An evaluation of videogame network architecture performance and security

Blake Bryant, Hossein Saiedian [*]

Information and TelecommunicationTechnology Center (ITTC) and Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS, USA

ARTICLE INFO

Keywords:
Network security
Videogames
Videogame networking
Network performance
Videogame security

ABSTRACT

The recent introduction of higher reward in videogame competitions is expected to motivate unethical players to pursue opportunities to gain unfair advantages while playing networked videogames. Networked videogames implement a variety of approaches to attain a balance between reliable data transfer and game performance. Certain aspects of these network approaches may be exploited by players to gain unfair advantage or degrade the gaming experience for others. This paper lays the conceptual groundwork for networked videogames by describing common network architectures that facilitate competitive videogame play. These networking concepts are then evaluated for susceptibility to potential exploits. Finally, three current gaming titles are selected as case studies, using the principles established within this paper, to evaluate the effects of client-side exploits, latency, and state synchronization, on competitive game play.

1. Introduction

A thriving economy of computer-based videogames has existed for well over four decades. However, a drastic spike in the popularity of highly competitive multiplayer games has propelled the industry into a new economic tier, on par with that of traditional spectator sports. Notable gaming titles, such as Fortnite, Defense of the Ancients 2 (DotA 2) and Counterstrike GO (CSGO), now offer tournament pots in the tens of millions of dollars [1]. Several content streamers on the videogame-oriented Twitch steaming service now earn seven figure salaries, with some boasting net worth in the tens of millions of dollars [2]. The astronomical rewards offered in such tournaments, or stream viewership, will naturally motivate unscrupulous individuals to seek opportunities to exploit the growing esports ecosystem. Within this context, the networking protocols used to facilitate online multiplayer games should be evaluated for their feasibility in a semi-trusted or contested environment.

This paper provides a brief survey of common approaches to networking in multiplayer videogames. This includes an overview of the TCP and UDP protocols and their impact on time sensitive network applications, such as videogames. This paper also briefly introduces some of the potential security concerns associated with network videogames, in the context of manipulating the flow of player control data. Specific focus is given to the “state saturation” attack, introduced as a novel technique, within this paper. Three videogame titles are analyzed as case studies depicting both contemporary examples of the networking approaches surveyed in this paper, as well as exploitable elements of their design that exhibit aspects of security concerns also introduced in this paper. Additionally, a novel attack, referred to as “state saturation”, is introduced as a security concern and demonstrated within the final case study of the paper.

Empirical evidence is collected through experimentation to measure game client network traffic generation and validate hypothesis pertaining to the “state saturation” attack. Additionally, anecdotal evidence is collected through analysis of social media to measure potential effects of suspected “state saturation” on player satisfaction during gameplay. This paper concludes with recommendations on modifications to game network models and protocols to retain the benefits of a technique referred to as “animation canceling” yet reduce the negative impacts it has on game performance and player satisfaction.

2. Background

This section provides a brief overview of standard networking approaches and their suitability for adoption in videogame communications, as well as current approaches to network protocol design in videogames.

2.1. Suitability of TCP vs UDP for videogame networking

Compulsory education in computer networking establishes a fundamental understanding of two overarching approaches to network protocol development: reliable and guaranteed delivery vs best-effort delivery protocols. The de facto protocol for reliable network data transfer

[*] Corresponding author.
E-mail address: saiedian@ku.edu (H. Saiedian).

https://doi.org/10.1016/j.comnet.2021.108128
Received 9 August 2020; Received in revised form 14 April 2021; Accepted 15 April 2021
Available online 20 April 2021

is the Transmission Control Protocol (TCP) published in RFC 793 circa 1981. The TCP protocol is extended by several additional RFCs. TCP related RFCs pertinent to this discussion follow: security improvements (RFC 1948), high-performance tweaks for large bandwidth–delay product networks (RFC 1323), congestion control (RFC 5681) and modifications to message acknowledgment options (RFC 2018). The User Datagram Protocol (UDP), published in RFC 768 circa 1980, is the primary alternative to the TCP protocol, and merely defines the absolute minimum amount of data and formatting required to send messages via internet protocol (IP) networks. Developers of networked applications are essentially limited to these two options when creating programs that communicate via the ubiquitous internet protocol (IP). It is possible to send data across a network without using TCP or UDP, with programs such as SOCAT; however, such a drastic approach would not be conducive to the development of cross-platform software intended for consumption by the general public leveraging commodity operating systems.

On the surface, TCP may appear to be the logical best choice for developing a secure and reliable networking protocol for videogames. It ensures data integrity by retransmitting messages that may have been lost in transit and provides mechanisms for ensuring all components of a message are reassembled in the proper order. The protocol implements a framework for addressing bandwidth contention between multiple hosts on a network resulting in equal share of resources and protects network infrastructure from being overwhelmed with traffic. Furthermore, TCP is a “connection oriented” protocol that establishes a persistent relationship between two communicating parties through which future messages may be contextualized. All these features are desirable in a videogame network environment; however, the means through which these features are implemented results in undesirable performance degradation for real-time applications that are highly sensitive to latency. John Carmack, the author of the influential Quake engine, commented publicly on his challenges in implementing TCP based communications when developing one of the first internet based first person shooter titles [3].

Other works have evaluated the suitability of TCP based protocols in high-performance networks and identified challenges with core components of TCP which drastically impact bandwidth utilization or increase latency, namely the use of the Additive Increase Multiplicative Decrease (AIMD) approach to congestion control, the means by which congestion is identified, and the cumulative round trip time (RTT) associated with the TCP protocol’s dependence on acknowledgment messages [4–8].

In summary, the TCP protocol is unable to tolerate packet loss. This is a desirable quality in many applications but is not necessary in certain real-time applications such as networked videogames. Packet loss under TCP based protocols initiates two intrinsic components of TCP. First, TCP assumes that all packet loss is indicative of network congestion and immediately implements congestion avoidance algorithms. There are several variations of these algorithms; however, one constant is that they all result in an immediate drastic decrease in bandwidth utilization followed by a much slower bandwidth increase to return to previous utilization levels. This network oscillation not only degrades performance in real-time applications, but may also be exploited to provide unequal bandwidth to different hosts, as was outlined in Li et al. [7]. The second issue associated with lost packets is the requirement to resend lost packets after detecting the loss. At a minimum, this re-transmission will incur an additional round trip time (RTT) to reach the recipient and have an associated ACK message routed back to the sender. In practice, the re-transmission penalty is greater than a single RTT as loss is either detected via a packet timeout or a total of four ACK messages for the packet sequence number proceeding the lost packet (the original ACK for the previous packet, followed by three duplicate ACK messages).

Another limitation of the TCP protocol is its approach to dealing with modern networks consisting of gigabit or higher capacity links extended over large distances. The TCP protocol contains fields within its packet headers to indicate the size of buffers between communicating hosts referred to as “window size”. Window size is used to indicate the amount of data that may be sent between communicating parties before receiving acknowledgment (ACK) messages and is intended to be used as a throttling mechanism. The original TCP specification outlined in RFC 793 used a 16-bit field to indicate window size, meaning hosts could send at most 64kb of data before receiving an ACK message. RFC 1323 addressed this issue by providing extension options to increase the window size of TCP up to 1 gigabyte. However, implementation of RFC 1323 extensions may be inconsistent across operating systems and networking devices along the path between hosts.

Another limitation of the TCP protocol is its approach to dealing with modern networks consisting of gigabit or higher capacity links extended over large distances. The TCP protocol contains fields within its packet headers to indicate the size of buffers between communicating hosts referred to as “window size”. Window size is used to indicate the amount of data that may be sent between communicating parties before receiving acknowledgment (ACK) messages and is intended to be used as a throttling mechanism. The original TCP specification outlined in RFC 793 used a 16-bit field to indicate window size, meaning hosts could send at most 64kb of data before receiving an ACK message. RFC 1323 addressed this issue by providing extension options to increase the window size of TCP up to 1 gigabyte. However, implementation of RFC 1323 extensions may be inconsistent across operating systems and networking devices along the path between hosts.

The final issue with TCP based protocols is that TCP is implemented within a computer operating system at the kernel level. As such, developers of videogames have little control over which specific TCP features are implemented or how they are implemented across various systems. This variation in TCP implementations could result in complications when attempting to implement deterministic timing between networked client and server videogaming platforms [9].

In the end, ironically, many of the features of TCP that make it desirable as a videogame protocol are implemented in a manner which will drastically inhibit performance. Fiedler’s work indicates that as little as 200 ms RTT delay and 2% packet loss in a TCP connection is enough to render a videogame unplayable [9].

What is truly needed is something between the traditional TCP and UDP models of either guaranteed reliability or no reliability. Unfortunately, it is impossible to decouple desirable components of TCP in order to tailor them to suit videogame networking. As such, videogame developers are required to develop UDP based protocols, essentially from the ground up, in order to implement some TCP-like capabilities without incurring latency associated with TCP overhead. The primary means through which this may be accomplished is by accepting packet loss and adjusting how acknowledgment messages are sent and handled between hosts.

2.2. General approaches to videogame network protocols

Fiedler’s work provides an overview of the three dominant approaches toward encoding videogame data to synchronize physics engines between hosts [9]. The primary goal of each of these approaches is to synchronize representations of the virtual environment between remote systems participating in the game simulation. The three approaches are: deterministic lockstep, snapshot interpolation and state synchronization. Each of these approaches impact the bandwidth requirements for synchronizing physics engines between hosts, as well as their ability to tolerate network latency.

2.2.1. Deterministic lock-step networking

The deterministic lockstep approach requires the least amount of data as it merely passes user input instructions to a remote host running an identical simulation synchronized with the sending hosts. However, this smaller packet size comes at the expense of an increase in both server and client side processing. One critical limitation of

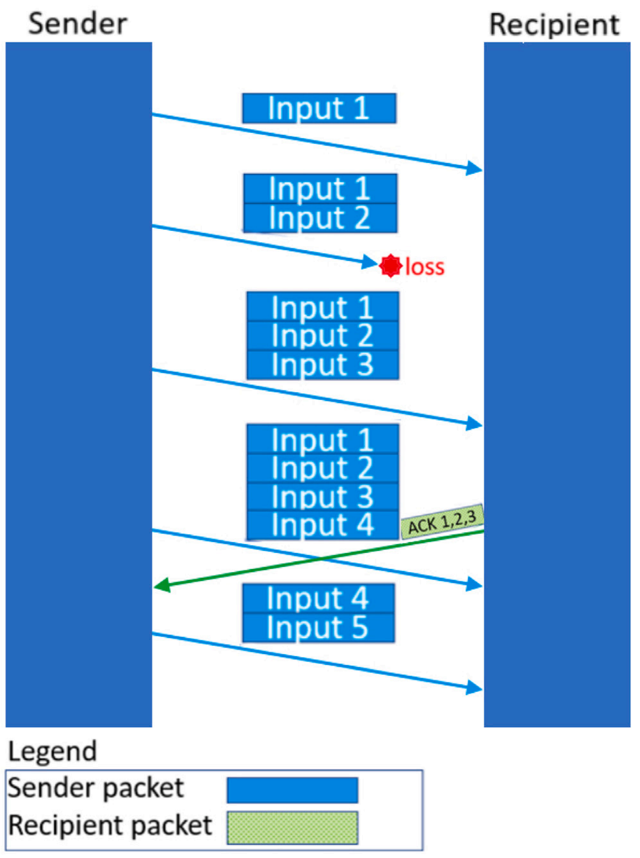


Fig. 1. UDP-based deterministic lock-step networking approach.

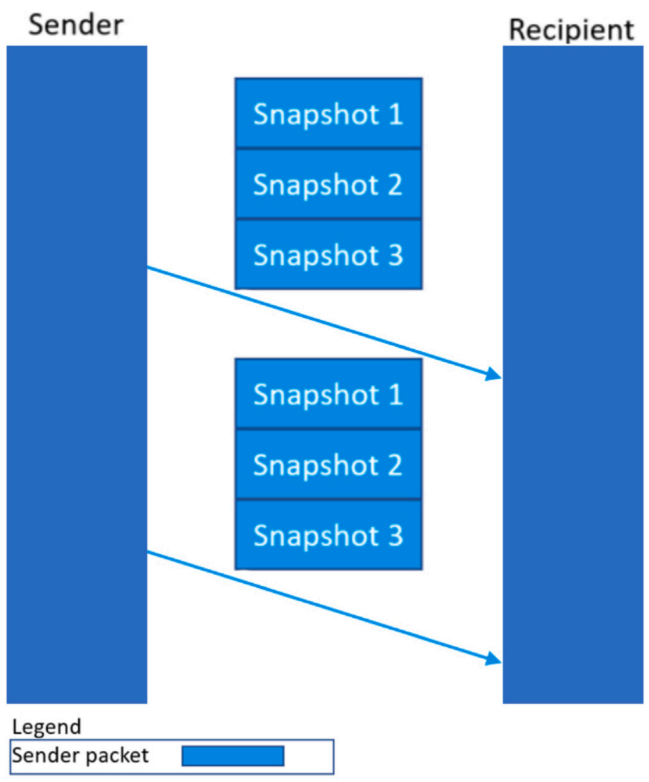


Fig. 2. UDP-based snapshot interpolation networking approach.

this approach is that the extreme processing demands placed on the server drastically limits the number of clients that may participate in networked games. According to Fiedler, the recommended maximum number of connections is limited to four clients to one server [9]. The Unreal Engine documentation also cites lack of persistence, lack of player scalability and lack of frame rate scalability as critical limitations of this approach [10]. Additionally, this approach requires games to be constrained to identical platforms due to difficulty in maintaining a deterministic state between different operating systems, hardware and compilers. Fig. 1 depicts this approach to implementing reliability via the unreliable UDP protocol.

The deterministic lock-step approach achieves reliable data transmission by requiring the client to send consecutive packets containing all unacknowledged user inputs. This effectively communicates the client packet buffer to the server until it is acknowledged, at which point, the client buffer is reduced to merely unacknowledged inputs. However, unlike the function of TCP acknowledgment packets, the client is not throttled by the lack of server acknowledgments. The advantage of this approach over TCP is obvious in that the client may continue to update its buffer and the server without incurring an additional round trip time (RTT) delay. It is also worth noting that simulations running on the server must wait until input instructions are received from the client in order to proceed with the simulation. This is one of the key differences between this approach and state synchronization discussed later.

2.2.2. *Snapshot interpolation*

The snapshot interpolation technique, depicted in Fig. 2, adopts a slightly different approach and sends a representation of the current state of the simulation between hosts. Typically, the server sends only the objects a client must draw in order to depict a frame representing the game world, then the client processes user input commands and sends the newly rendered state of objects previously sent from the server. This exchange will undoubtedly require more information than simply replicating user inputs. As such, bandwidth requirements increase, and games are less tolerant of network latency. Furthermore, bandwidth requirements with snapshot interpolation are dependent upon the number of game objects that must be updated making this a poor choice for simulations with many object interactions.

Snapshot interpolation overcomes the unreliable nature of UDP by assuming that some packets will occasionally be lost, but gaps in data may be addressed by mathematically inferring a trend line between the two states received. For instance, if the recipient receives snapshots for state 1 and state 3, but not state 2, the recipient can simulate a transition between the two states received and avoid waiting to receive state 2.

An unfortunate side effect of this approach is the need to buffer multiple state snapshots prior to sending packets. This buffering introduces additional delay on top of network delay and is dependent upon the simulation framerate in the form of the equation latency=R/B; where R represents the framerate and B represents the number of packets to be buffered. Typically, networked games operate on a standard 60 frames per second refresh rate, meaning a buffer of three snapshots would incur a delay of 50 ms in addition to the network latency. Fig. 3 depicts the concept of buffering snapshots in order to prevent data gaps due to lost packets.

Despite its limitations, snapshot interpolation overcomes the issues associated with the deterministic lockstep approach, namely head-of-line blocking waiting on packets to arrive and requirements for strict control over client and sender hardware/software configurations.

2.2.3. *State synchronization*

The state synchronization approach effectively builds upon the advances made from the previous two techniques. Namely, simulations

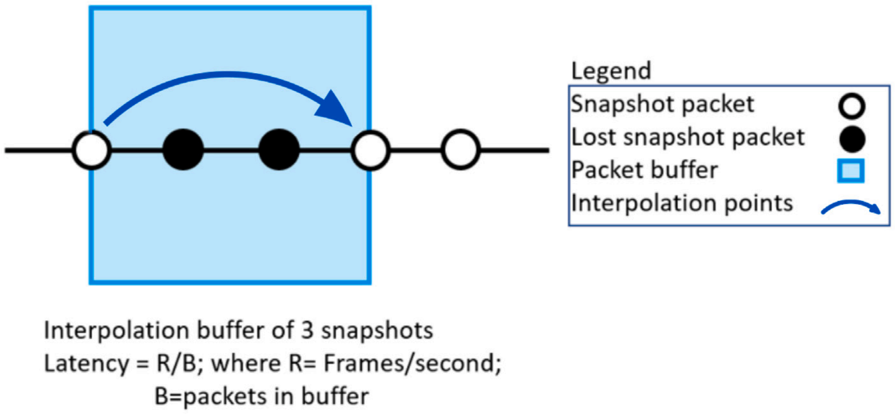


Fig. 3. Snapshot interpolation buffering.

are run on both client and server systems which render simulations in response to inputs generated on the client and are sent to the server. However, unlike the deterministic lock-step approach, the server does not wait for inputs if they do not arrive, but rather predicts what would happen based on previous inputs and reconciles differences if/when client inputs arrive. Additionally, the client-side rendering is subject to modifications based on state updates sent from the server's representation of the game world. However, unlike snapshot interpolation, wherein the entire game world is sent, only a subset of object updates must be sent to the client for rendering. Which objects are sent in updates is based on a priority queue. Fiedler devised a technique known as a priority accumulator that tracks the priority of objects persistently between frames and accrues additional priority based on how stale the object is; meaning objects that have not been sent recently will naturally move to the front of the queue and ensure they are eventually sent [9]. Fig. 4 depicts this behavior.

The priority accumulator essentially maintains the status of objects that must be rendered in each frame whilst applying cumulative weights to objects that are most impactful on the gaming experience. For instance, player character objects will have a high priority, projectiles in shooting games will have the highest priority (even above player characters), and static objects will have the lowest priority.

The priority accumulator is an effective bandwidth reduction tool in that it may selectively send only the most important objects, and defer less critical objects for future updates. This also allows this technique to adhere to strict bandwidth limitations imposed by the game engine, potentially to ensure fairness amongst clients.

Also, it is worth noting, that the cumulative nature of the aptly named priority accumulator ensures that even low priority objects will be updated eventually, as their previous priority persists and accrues additive priority between frames until they are sent in a packet.

3. Industrial application of networking approaches

The following section provides examples of videogames that have implemented the networking approaches introduced in 2.2.

3.1. *Deterministic lock-step example — DOOM to starcraft II*

The first person shooter DOOM, circa 1994, was based upon a peer-to-peer networking model implementing the deterministic lock-step approach [11]. The DOOM network implementation consisted of multiple hosts running identical software serving as both client and server systems, with no logical "authoritative" system within the network. Each peer node would sample user input 35 times a second, package said input into a data structure called a "tick" and sent ticks to all other peer systems on the network. The simulation could not progress until all clients had received all ticks from every other peer.

Network latency under this model was addressed by buffering inputs until all outstanding ticks were received from peers. Though the simulation appeared to be stalled, once the final tick message was received, multiple buffered ticks could be processed in rapid succession. Unfortunately, as simulation progression was dependent upon tick delivery to all peer nodes, game responsiveness to user input was directly tied to RTT delay within the network. This made the quality of game play highly dependent upon both the number of peers and the reliability of the underlying network infrastructure.

The synchronous nature of this approach also introduced several challenges. Pairing players with one another was a complicated task, as joining an existing match in progress was not possible. Furthermore, slight variations in client hardware was suspected of causing floating point drift errors, such as different graphics card manufactures generating slightly different renderings of game objects. The end result being that hitboxes and player positioning were not consistent between different player realities, ultimately degrading the quality of the game experience.

The peer-to-peer approach was also criticized for being vulnerable to client-side cheats. Though a client could not modify the collective state directly, they could alter their perception of reality by modifying source files on their local game engine. The end result was the ability to remove visualization inhibitors, such as seeing through walls, or other obstacles intended to obfuscate player movement [11].

Despite the limitations in this approach, some modern games still leverage peer-to-peer deterministic lock step approaches. Blizzard's Starcraft II real-time-strategy (RTS) offering still leverages a peer-to-peer architecture; however, Blizzard made slight changes to the routing architecture to implement an intermediary server to store and forward packets between peers [12,13]. Despite the introduction of a client–server implementation, the underlying protocols, packet structure, and simulation behavior remains a peer-to-peer deterministic approach in current implementations of the Starcraft II game.

3.2. *Snapshot interpolation example — Counterstrike*

The first person shooter Counterstrike Global Offensive leverages the snapshot interpolation approach [14]. An example exchange leveraging this model is depicted in Fig. 5. Rather than relay user input between peers, the Valve architecture relays user input from clients to a central server which renders the inputs into its representation of the simulation world. Snapshots of the server representation are redistributed to all clients at a rate determined by configurations specific to each game. The Counterstrike game operates on a tick rate of 66

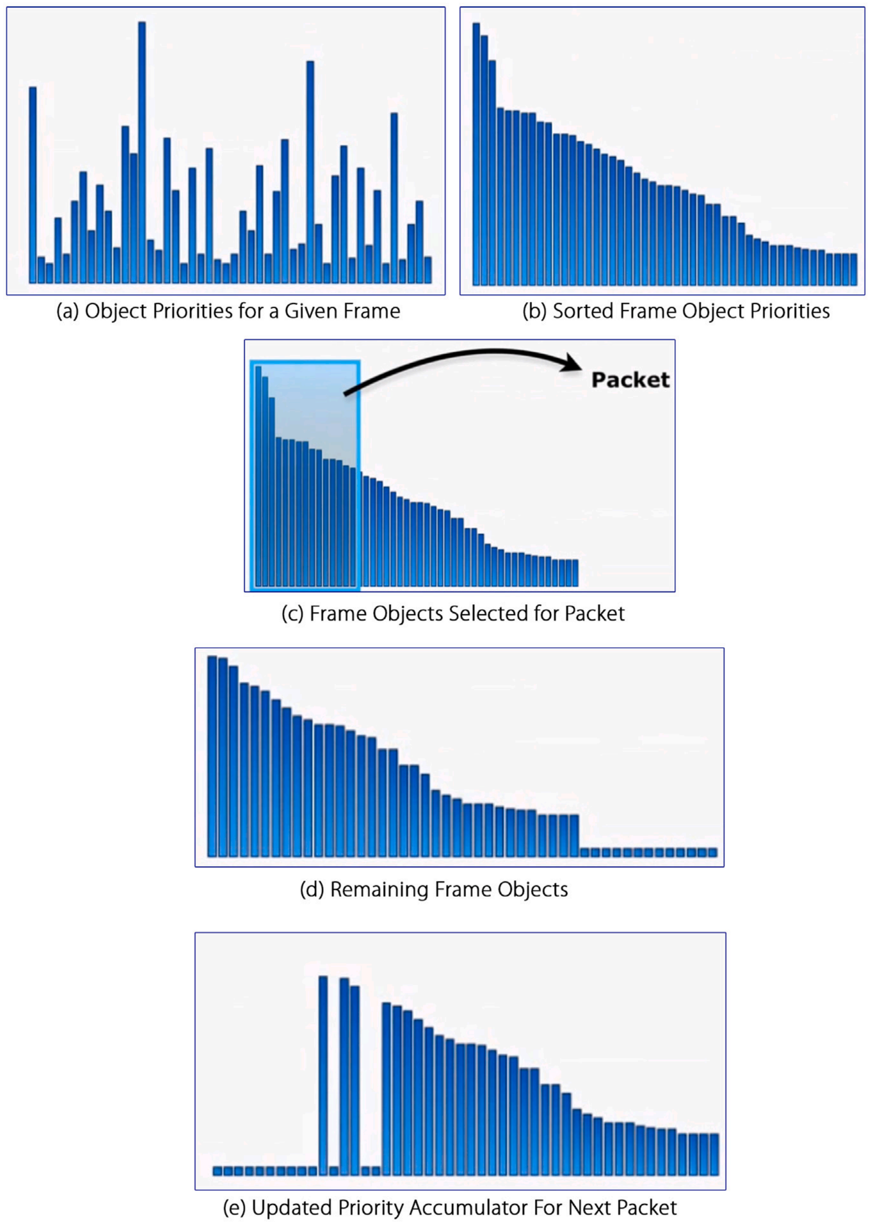


Fig. 4. Fiedler's priority accumulator [9]

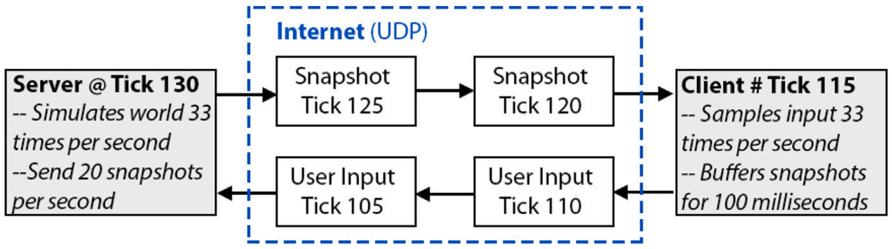

Internet (UDP)
Server @ Tick 130
-- Simulates world 33 times per second
--Send 20 snapshots per second
Snapshot Tick 125
Snapshot Tick 120
User Input Tick 105
User Input Tick 110
Client # Tick 115
-- Samples input 33 times per second
-- Buffers snapshots for 100 milliseconds

Fig. 5. Valve multiplayer networking model [15].

simulations per second, while Fig. 5 reflects a slower tick rate of 33 per second.

Comparative analysis of the Valve snapshot approach and Blizzard's deterministic lock-step approach indicate that snapshot traffic is prone to higher bandwidth consumption bursts and produces less consistent traffic patterns [16]. This variation in traffic patterns is expected as snapshot-based message schemes will produce varying data requirements as the state of the simulation world varies. Claypool et al. noticed

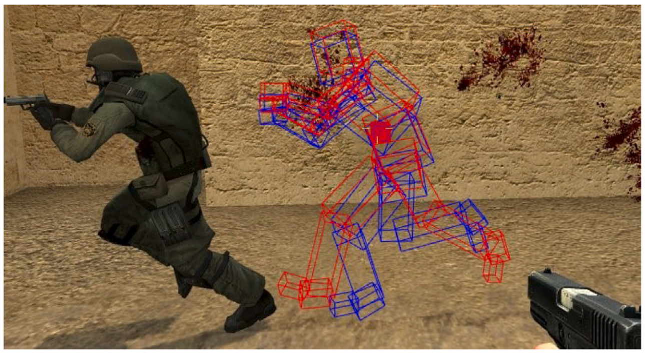


Fig. 6. Valve snapshot lag compensation [15]. (For clarity, a color version of this figure is available on the web version of this article).

large variations in Counterstrike traffic following the introduction of a new player or the virtual death of an existing player object. Additionally, the lack of client level synchronicity in the Valve network implementation enabled players to join or leave existing game sessions in an ad hoc manner. This fluctuation in client membership may have also attributed to variations in the Counterstrike traffic pattern.

The Valve snapshot implementation broadcasts the world state amongst all clients participating in a simulation [14]. This results in all clients maintaining a common representation of the game world, including objects that should not be known based on a player's current virtual perspective. Sharing a common representation of the virtual world introduces the possibility of client-side hacks to expose data that should be hidden from players, like those seen in peer-to-peer architectures. Valve has attempted alleviate the prospect of client-side exploits by implementing a specialized version of malware detection via its proprietary Valve Anti-Cheat System (VAC).

The asynchronous nature of the Valve snapshot implementation also introduces some unique gameplay issues discussed in [11,15]. The Valve snapshot interpolation technique leverages local prediction on clients to process input commands prior to receiving state updates from the authoritative server. This phenomenon is depicted in Fig. 5 as the difference in tick IDs between the client and server, with the server operating on more recent snapshot ticks than the client. Fig. 6 depicts the impact this lag in state processing has on a client.

Fig. 6 depicts the current client's perspective as red mesh hitboxes and the server's state perspective as the sprite of the soldier to the left. The blue mesh hitbox represents the server's attempt to "roll back" game time and coordinate the player's previous input with how it would have been interpreted when the input was transmitted. Despite advances in client and server prediction models, it is possible for measurement errors to result in simulated projectiles "trailing" targets or potentially missing targets all together.

3.3. State synchronization example — Tribes to fortnite

The multiplayer online game Starsiege Tribes, circa 1998, was one of the first games to implement the state synchronization approach [17]. The Tribes networking model was built around three distinct components depicted in Fig. 7. The simulation layer is responsible for maintaining the tick rate of the simulation, determining which objects to update in state messages and conducting client prediction. The Stream layer is responsible for exchanging datagrams between the host and server nodes and is responsible for compression, encoding, and selection of quality of service. The connection layer is responsible for implementing the different quality of service guarantees determined by the stream layer, essentially by managing whether packets must be acknowledged (via the connection manager) or are sent via best effort delivery (through the platform packet module). The stream layer of the Tribes model is the most pertinent to this paper and will be elaborated upon further.

The stream layer is comprised of several sub-components called managers. Each manager is responsible for determining what data should be inserted into packets to be sent via the connection layer. These managers are effectively used to establish the priority and means by which data is transferred and takes the place of the priority accumulator described by Fiedler.

The Ghost Manager is responsible for generating state data based on object scoping established by simulation layer. Object scoping refers to the process of determining which subset of world objects should be selected as "dynamic" elements in the simulation, such as player characters, or things players can interact with. "Ghosted" objects will maintain a state matrix representing sub-components that have been modified since the last state update had been sent via a state mask. A state mask is essentially a dirty bit approach to identifying changes in the ghost object properties. Ghost Manager state data is transmitted to clients similar to how snapshots operated in the snapshot interpolation approach; however, ghost data only consists of a subset of objects within the simulated world. Additionally, ghost snapshots are considered volatile data and only the most recent snapshot will be processed, meaning reliable delivery is less important than timely delivery.

The Move Manager is responsible for packaging player input data to be sent to the simulation layer running on the server, similar to how input was delivered in the deterministic lock-step approach. Input data is considered the highest priority data and intended to be sent as soon as possible. However, surprisingly, input data is not sent using an acknowledgment scheme controlled by the connection manager. Instead, reliability of input data is achieved by sending input data in every packet from the client and transmitting each user input into three consecutive packets and maintaining a sliding window of outstanding moves. User input instructions are removed from the sliding window buffer once they have reflected in state updates received via the Ghost Manager.

The Datablock Manager is responsible for providing guaranteed delivery of relatively static objects. This may be thought of as the initial loading time when data stored in environment variables must be passed to clients in order to properly render the simulation on client machines.

The Tribes approach to establishing a dedicated stream layer is unique in that it provides the capability to establish an upper bound on bandwidth. Since the stream layer is responsible for reconciling data priority scheduling as well as bit packing for transmission, it can send priority data via available bandwidth and delay, or fragment, transmission of less critical data across subsequent packets. This property makes the Tribes state synchronization technique one of the most appealing approaches from bandwidth and latency perspectives.

Another benefit of the Tribes approach to state synchronization is that partial state updates distributed via the Ghost Manager effectively prevents exposure of unintended state data to players via manipulation of client side binaries.

The popular game Fortnite is based upon the Unreal game engine which also operates under a state synchronization networking approach [10]. Many of the same approaches to data exchanges between clients and servers were preserved in the Unreal engine's implementation of this approach, though obviously terminology for concepts were changed when implemented by a different development team. The Unreal engine also uses a "dirty bit" scheme to identify subsets of game objects to be included in state updates sent to clients. However, it is worth noting that the Unreal engine specifically uses player perspective to determine object scoping, meaning objects within a player's view, or able to effect a player, should be the only things sent to said player in updates. This approach offers the benefits of reduced bandwidth overhead in updating unnecessary objects as well as preventing leakage of other player state data.

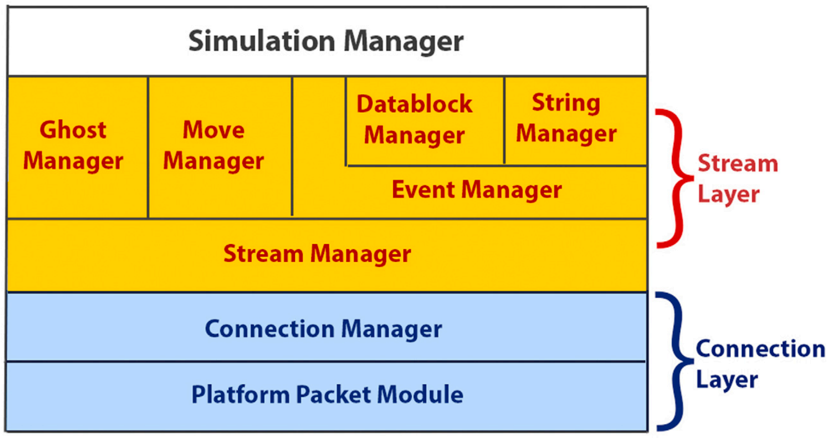


Fig. 7. Tribes network model components [17].

4. Security concerns with networked videogames

The background section of this paper described the unique networking challenges and approaches videogame developers apply to delivering simulated experiences. This section evaluates potential approaches attackers may use to exploit vulnerabilities in networked videogame implementations.

4.1. Client-side exploits

Documentation associated with each of the example implementations of the deterministic lock-step and state synchronization approaches cited concerns with unethical players attempting to modify client-side binaries to gain an unfair advantage during game play [3,11,15,18]. One of the critical weaknesses of these two approaches is that they rely on broadcasting complete state data to all clients. As such, unethical players may modify their systems to remove intended restrictions on client interpretation of state data.

For example, the real time strategy game Starcraft II, and other similar titles, implement a “fog of war” effect, wherein players may only see a certain portion of the map that would be visible to units within their control. However, client-side modifications to the way in which the player’s version of the game renders state data could result in disclosure of the entire map, giving one player a marked advantage over others. First-person-shooter client-side exploits could include aimbots or trigger bots that generate “hit” notifications on behalf of a player once they are within range of a competitor’s character.

4.2. Timing exploits

Snapshot interpolation and state synchronization approaches implement client-side prediction models and lag compensating techniques that allow for the queuing of player inputs and replaying of past server-side state models to detect “would be” collisions from delayed player input [10,14]. This feature may be exploited by modifying the timestamp associated with player input updates, or merely adjusting the client clock time during play. This was specifically cited as a concern within the Epic Unreal Engine networking documentation [10]. Unfortunately, as all of the networking approaches described in this paper rely on best effort delivery, precise timing is impossible due to the possibility of lost packets.

4.3. State saturation

The state synchronization approach may be susceptible to a novel exploit presented in this paper known as “state saturation”. State saturation refers to the process of intentionally stimulating an excessive amount of objects to be rendered within the scope of another player’s simulation. As state synchronization approaches operate under the premise that only a portion of simulation world objects must be rendered by clients, and therefore consume bandwidth in periodic updates, and further more objects are determined to be in scope based on the visual perspective of a player’s virtual character, an unethical player could attempt to force a disproportionately larger number of objects to be generated on a victims machine than on their own.

A notional scenario wherein this could occur would be something akin to causing an environmental avalanche, or detonating several graphically intensive explosions within a victim’s view, while simultaneously out of the attackers view. In theory, the rapid introduction of new objects that must be rendered by the victim would adversely effect priority scheduling for the victim’s state updates and either exhaust their bandwidth available to process moves, or inhibit their ability to render the attacker’s actions in a timely manner, effectively reducing their available reaction time. This could give one player a marked advantage over another during a competition, but would be highly dependent upon the game’s state scoping model and potentially simulation specific environmental factors.

4.4. Volumetric denial of service

All network based game models are susceptible to bandwidth exhaustion associated with volumetric denial of service attacks. However, surprisingly, the oldest networking approach, using strict determinism, may be the best approach for thwarting such attacks. Deterministic lock-step networking approaches force the simulation to halt and await input from each client prior to progressing. An attacker attempting to exhaust a victim’s bandwidth to introduce lag into their simulation could unintentionally introduce lag into their own experience as well. However, the effectiveness of this technique will vary between game implementations, depending on whether the networking model enforces strict determinism or allows for input loss.

5. Case studies

The following section of this paper provides contemporary examples of videogames using networking approaches described previously within Section 3, combined with potentially exploitable security concerns described in Section 4. Each of these examples depict scenarios where players familiar with the underlying network model, and exploitable features, may leverage both to gain unfair advantage during game play.

5.1. Case study I- risk of rain 2 - client-side exploitation

Risk of Rain 2 is a first-person shooter rogue-like game designed to support up to four players in simultaneous cooperative gameplay. The primary objective in the game is to discover a special teleporter in each level that unlocks a boss fight and leads to a series of new level maps. Waves of procedurally generated adversaries, of increasing power and number, appear to pose challenges to players during their search for the teleporter. Additionally, players may find several randomly generated power-up objects that improve the power and/or behavior of their characters during an attempt to beat the game. If all player characters die, then the game restarts and all powers gained during a previous attempt are lost. Variability in powers acquired by players between game runs is the main draw for the player base of this game.

A short-lived feature introduced into game play between August 04 to August 14 2020, involved the combination of two power-up items, one called “forgive me please” and the other called the “soulbound catalyst” [19]. The “forgive me please” power-up implemented an ability that immediately triggers any effects that would normally occur once an enemy was killed; while the “soulbound catalyst” reduced the cooldown timer of an item once something was killed. Surprisingly, the “forgive me please” item was considered as both an usable item and a kill-able object.

Conditions suitable for rapid successive uses of the “forgive me please” item were easily attained by killing an enemy, thus implementing the “soulbound catalyst” cooldown reduction, then using the “forgive me please” item, which itself counted as a kill and triggered another “soulbound catalyst” cooldown reduction, allowing for an immediate reapplication of the “forgive me please” item. The end result of combining these two objects was that the intended throttling mechanism, e.g. a cooldown timer, of the “forgive me please” item would approach zero after it was used two consecutive times [20].

To further compound this bug, a third power-up item called “gesture of the drowned” forces usable items to automatically activate once their cooldown timer expires. As previously demonstrated, the cooldown timer of the “forgive me please” item would effectively reach zero almost instantaneously when combined with the “soulbound catalyst” item. Therefore, players could easily initiate a chain reaction resulting in the destruction of all enemies currently in existence.

The cooldown timer was originally intended to prevent players from using items in rapid succession, which this combination violated. Furthermore, potential on-kill effects that could be coupled with the “forgive me please” item included damage producing explosive effects or player healing effects, both of which generated visual objects to be drawn on player’s screens. The rapid generation of visual objects resulted in extreme degradation of system performance, to the point of either reducing game frame rates to unacceptable levels or causing players to be booted from the game due to system crashes.

The Risk of Rain 2 peer-to-peer network model implemented a deterministic lock step approach. Therefore, drastic reduction of one player’s game performance resulted in uniform performance reductions across all players, as each player depended on updates from all remaining players before progressing through the simulation. Additionally, the nature of the game focused on cooperative, rather than adversarial gameplay. As such, the performance impacts of the “forgive me please” bug did not introduce a competitive advantage per say, other than allowing a small population of players to beat the final boss of the game, or garner speed run achievements, with a trivial level of difficulty.

The client-side exploit illustrated in this example demonstrates the potential negative effects of bypassing client-side throttling mechanisms, e.g. cooldown timers, and generating an excessive amount of objects for clients to render. The ability to cause network lag by overwhelming a single node is a hallmark sign of the deterministic lock-step networking approach; however, the ability to bypass client-side throttling mechanisms may also be detrimental to other networking approaches, as will be illustrated by case study III in Section 5.3 of this paper.

5.2. Case study II — dead by daylight — timing exploitation

Dead by Daylight is a player versus player survival horror video game designed to support five players. The viewpoint of the game is a hybrid first-person and third-person perspective, split between two competing teams. One team consists of a sole player, known as the “killer”, while the other team is comprised of four players known as “survivors”. The “killer” team views the game from a first-person perspective, while the “survivors” view the game from a third-person perspective.

The gameplay is competitive in nature and requires the sole “killer” to eliminate each of the “survivors”, while “survivors” attempt to solve puzzles toward the goal of eventually escaping from a maze-like level. Players on both teams may perform actions that improve their score and generate in-game currency which they may use to purchase desirable upgrades.

From a networking perspective, Dead by Daylight implements a state synchronization-based hybrid peer-to-peer and client–server model. Players establish ad-hoc client and server relationships, with one player randomly serving as the authoritative server, while all other players act as clients. The player that is selected as the authoritative server is afforded a distinct advantage over other players in that latency mitigation techniques defer to the server to dispute differences between its predicted simulation state and updates provided by other players.

Savvy players have discovered that intentionally delaying the receipt of update messages from “client” players allow for the “server” player to manipulate the game state in strategically significant ways during game play. The practice of manipulating network connectivity has become so commonplace that players have created devices known as “lag switches” that include physical toggles designed to serve and restore physical network connections at will [21].

For an example as to how this may benefit a player, if a “killer” player’s system is acting as the game server and they identify a “survivor” player is near, they may temporarily disable their network connection, then move to the “survivor” player and defeat them before reestablishing their network connection. As the “killer” player system served to maintain the authoritative game state, any actions attempted to be transmitted in updates by the “survivor” character to evade or counter the “killer’s” actions would be discarded. The end result is the victim “survivor” player merely receives an update message indicating their character was killed, from what appears to be a teleporting “killer”.

Likewise, a “survivor” player, acting as the authoritative server, may move to one of the necessary puzzle objects required to win the challenge, start solving the puzzle, then disable their network connection and move away from the puzzle object. Once the “survivor” reinstates their network connection, their current position, which is now no longer stationary in front of the puzzle object, will update all client systems, creating the appearance that the “survivor” teleported. A “killer” player observing this would see a “survivor” instantly teleporting across the screen. A savvy “survivor” player could maximize this technique by strategically “teleporting” between objects that obscure a “killer’s” view.

The ability to control client access to the authoritative server was trivial to attain in this example; however, despite the ease with which this exploit may be conducted, this scenario provides insight to the potential dangers of preventing select game clients from providing timely updates to communal authoritative severs. The ability to delay player update messages sent to the authoritative server is a key component of the exploit exhibited by case study III in Section 5.3 of this paper. Specifically, the ability to delay transmission of select client updates being sent to the authoritative game server may afford an asymmetric competitive advantage to players controlling network traffic flow.

5.3. Case study III — the elder scrolls online — state saturation

The Elder Scrolls Online is a AAA massive multiplayer online role-playing game which claims to have accrued more than 15 million players between its launch in 2014 and January of 2020 [22]. The Elder Scrolls Online caters to multiple audiences, ranging from players who are drawn to the social aspect of an online community, to players who are diehard competitors interested in player-versus-player (PVP) combat or achieving leader board status in player-versus-environment (PVE) content. This case study focuses primarily on players who ascribe to the competitive category. However, the social element of The Elder Scrolls Online's community provided a wealth of social media data via comments posted to the game's official message forum.

5.3.1. The elder scrolls online network model

Though the exact innerworkings of The Elder Scrolls Online network communication layer is unknown, Zenimax Online Systems publicly stated that early development of the game made use of the publicly available game engine known as the HeroEngine platform [23]. The official HeroEngine documentation outlines the inter server communication model used to send updates between its component servers [24]. Game client information is replicated to area servers which serve as load balancers for communications to the master game server. As players' characters traverse the game world, their character model information is replicated between area servers and handoff is performed during a brief loading screen presented to the game client and player.

HeroEngine documentation indicates that bandwidth optimization techniques, similar to those described in Fiedler's state synchronization approach, are employed to decrease the size of each replication message sent between game clients as well as between area proxy servers and the master game server. Area servers track player character position information to be implemented as a factor in determining spatial awareness and determining object priority for replication. Objects that are further away from a player character object will have a lower priority in replication and may be throttled to decrease replication bandwidth. Furthermore, replication messages are optimized with bit packing so only byte streams are sent between the client and servers with special bit offsets mapped to data fields in lookup tables on game servers. This drastically decreases bandwidth consumption but makes reverse engineering of the protocol problematic.

The bandwidth optimization techniques appear to be effective in setting hard limits on player client communication rates. Client data rates are limited to the maximum amount of information the client may communicate within a one second burst, not to exceed 40,960 bytes. Furthermore, the size of individual replication messages may not exceed 4096 bytes. This indicates that only 10 maximum size replication messages may be sent between the game client and the game servers in a single second.

Zenimax Online Systems may have implemented a custom game engine between 2012 and 2014, the period between their leasing of the HeroEngine and the release of The Elder Scrolls Online game, however analysis of pcap data collected via Wireshark indicates there are several similarities between the Zenimax Online Systems game engine and the HeroEngine. The Zenimax Online Systems engine implemented in The Elder Scrolls Online appears to implement several different servers that the game client communicates with, as indicated by the variety of IP addresses contained within their communication. Furthermore, these IP addresses change when a player's character transitions between different areas of the game. Additionally, the number of update messages sent between the client and the server increase in proportion to the number of dynamic game objects (such as other player characters) within proximity to the current game client-controlled character. The game engine used in The Elder Scrolls Online also makes exclusive use of the TCP protocol, like the HeroEngine.

5.3.2. Analysis of social media data

The Elder Scrolls Online exhibits a healthy social community of players who host content on private websites or stream content via outlets such as Twitch or Mixer, all of which benefit from ad revenue associated with viewership. Actual subscriber numbers are not known to the public; however, active player counts may be extracted from statistics contained within the Steam content delivery service. Steamcharts.com reports concurrent player count typically fluctuates between 20,000 to 30,000 players online at a given time, with a peak volume of 49,000 concurrent players [25].

A web scraping program was developed to gather data from the official user forums and support traditional SQL based relational queries to draw inferences from forum posts [26]. A total of 3.2 million messages, representing 164,000 distinct conversations, were analyzed. References to player action priority or mentions of a client-side exploit were of particular interest for this study.

Analysis of social media data indicates the Elder Scrolls Online fosters an active and vocal competitive player-versus-player (PVP) community. 6.75% of all forum topics analyzed were dedicated to discussion of PVP gameplay and 35.2% of all discussion topics containing at least one message referring to PVP activities. 34.03% of discussions contain a reference to game fairness, balance, or relative difficulty adjustments for player characters. The player community also comment on tactics or techniques used to improve performance within the game indicated by 15.6% of discussion topics containing at least one reference to the damage-per-second (DPS) statistic used to measure player performance within the game. The community also comments on the game's performance, with 3.2% of discussions referencing game performance or network lag.

DPS is often determined through skillful execution of player actions within an optimal sequence, referred to as a parse or rotation. References to this prioritized action sequence occurs in 6.2% of discussions, and a concept known as "animation canceling", which will be expanded upon later in this paper, occurs in 3.98% of discussions.

Analysis of message composition for specific topics pertaining to DPS performance indicates that "rotation" or "parse" appears in 16.79% of messages that also contain the term "DPS". Furthermore, 10.96% of messages that reference "parse" or "rotation" also reference animation canceling. Interestingly, 9.77% of topics pertaining to animation canceling also contain at least one reference to "bugs" in the Elder Scrolls code. Forum discussions such as these ultimately exposed the presence of exploitable features within The Elder Scrolls Online game client that may be used by players to bypass player input throttling mechanisms covered in the following section of this paper.

5.3.3. Exploiting client side validation — animation canceling and weaving

The previous section of this paper introduced references to the terms "animation canceling" and "weaving" in the context of an exploit within The Elder Scrolls Online game. Despite the low percentage of message volume within the forums dedicated to these topics, many expert and competitive players in The Elder Scrolls Online community laud this technique as one of the most important elements of high-performance game play. Unfortunately, Zenimax Online Systems has not released an official statement explaining how their network code and client-side validation operate in concert. There are some clues hidden within the developer patch notes, but there are also several references and explanations provided by the player community.

One of the earliest references to the concept of animation canceling, within the official Elder Scrolls Online forums, dates back to May 2015 [27], which in itself references a quoted response from the game developer in 2014 stating that "[animation canceling is] not exactly intended, but not an exploit. It is one of those things we did not really expect". An even earlier reference to animation canceling may be found within the web-based guides developed by a prominent YouTuber named Deltia, who also hosts the website Deltia's Gaming. Deltia boasts over 116,000 subscribers to their YouTube channel, and

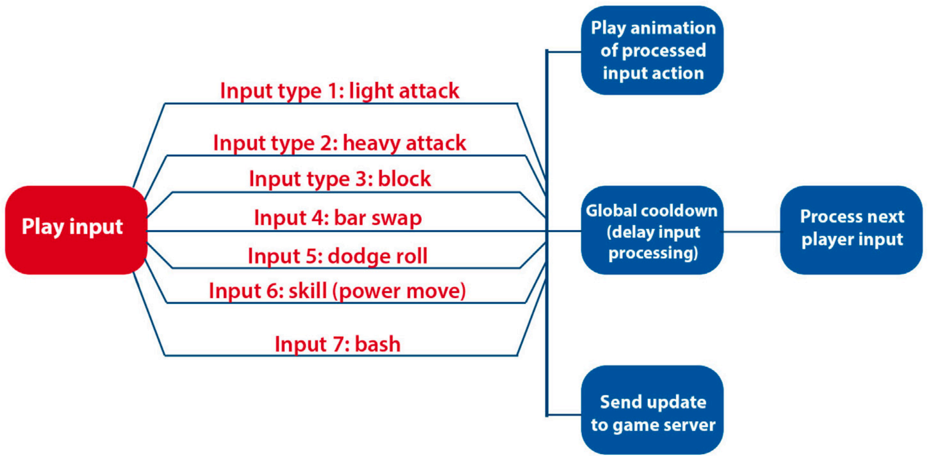

Play input
Input type 1: light attack
Input type 2: heavy attack
Input type 3: block
Input 4: bar swap
Input 5: dodge roll
Input 6: skill (power move)
Input 7: bash
Play animation of processed input action
Global cooldown (delay input processing)
Process next player input
Send update to game server

Fig. 8. Elder scrolls online player input processing.

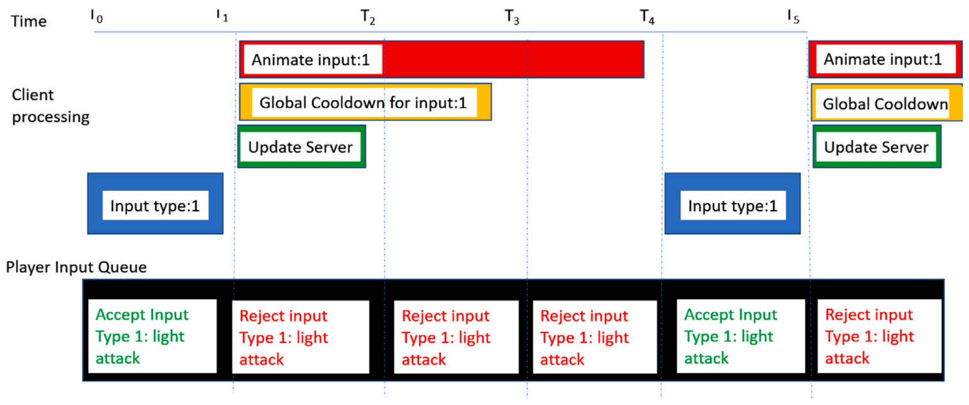

Time
T0
T1
T2
T3
T4
T5
Client processing
Animate input:1
Global Cooldown for input:1
Update Server
Input type:1
Animate input:1
Global Cooldown
Update Server
Input type:1
Player Input Queue
Accept Input Type 1: light attack
Reject input Type 1: light attack
Reject input Type 1: light attack
Reject input Type 1: light attack
Accept Input Type 1: light attack
Reject input Type 1: light attack

Fig. 9. Scenario 1: Elder scrolls online intended input throttling behavior.

over 85,000 views of their animation canceling video [28]. Deltia's guide on animation canceling was released in October of 2014, just six months after the release of The Elder Scrolls Online in April of 2014. Another popular authority on competitive gameplay with the Elder Scrolls Online community, who goes by the name of Alcast, published a recent animation canceling guide in August of 2019 on his private webpage [29]. Additionally, several YouTube personalities regularly post periodic videos of their character's damage potential between different patch versions of the Elder Scrolls Online game. These videos provide objective data points for comparing relative performance impacts that developer modifications introduce on gameplay as the software continues to evolve. One such YouTube personality, who goes by the name of Arzyel, claims that all of their performance evaluations make heavy use of animation canceling, and even links viewers to a video explaining how to perform animation canceling [30].

As stated previously, there are no official documents provided by Zenimax Online Systems that outline the actual user input processing design implemented within their game client. However, user feedback and community developed instructional material, such as those referenced in the previous paragraph, provided insight of user input processing that inspired the development of Figs. 8 through 12. Fig. 8 depicts the order of operations conducted when processing player input. First, the player input is received by the client and assessed to be associated with one of several types of actions. Seven types of actions are depicted within the figure, each associated with a different player-initiated input. Some forum posts indicate that there may be additional actions, not associated with user input, such as procedurally generated effects based upon chance. However, the model depicted in this paper is specifically limited to player provided input as this is critical to evaluating the competitive nature of the exploit. Procedurally generated effects are assumed to be attributed to one of these seven inputs for simplicity. The third step in the player input processing model entails three simultaneous actions initiated after user input has been validated and accepted by the client. These three actions are, playing an animation sequence of the accepted input, initiating a "global cooldown" timer, and sending an update of actions performed to the game server. The final step in this model is the acceptance of new input from the player.

It is important to note that each of these seven possible user input types are assigned a different level of priority, which will prove to be an important factor when discussing input throttling. Fig. 9 reflects the intended throttling mechanism to be used for player provided input, which shall be referred to as 'scenario 1'. If the player attempts to submit additional inputs of the same type, and therefore same priority, these additional inputs will not be accepted until all actions described in the preceding paragraph have been completed, which occurs at time T_4 in Fig. 9. Note that if a player repeatedly attempts to submit multiple consecutive actions of the same type prior to completion of the steps outlined in the previous paragraph, then the input will be rejected and not processed by the game engine.

Another scenario exists, however, wherein a player begins to initiate an action, but then chooses to react to another player or other

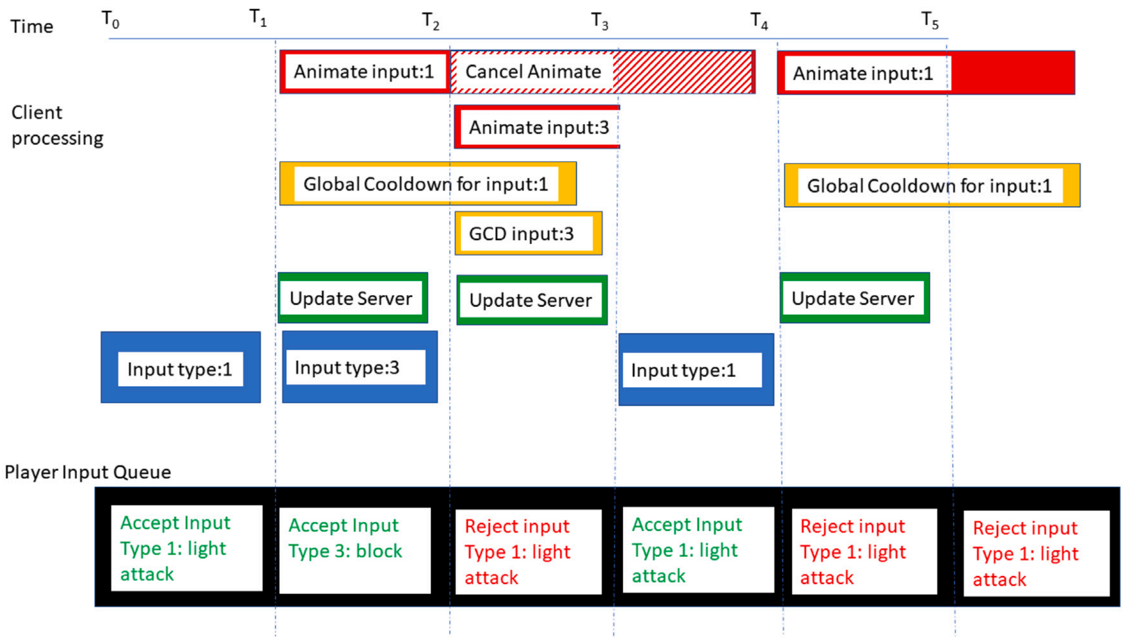

Time
T0 T1 T2 T3 T4 T5
Client processing
Animate input:1
Cancel Animate
Animate input:1
Animate input:3
Global Cooldown for input:1
Global Cooldown for input:1
GCD input:3
Update Server
Update Server
Update Server
Input type:1
Input type:3
Input type:1
Player Input Queue
Accept Input Type 1: light attack
Accept Input Type 3: block
Reject input Type 1: light attack
Accept Input Type 1: light attack
Reject input Type 1: light attack
Reject input Type 1: light attack

Fig. 10. Scenario 2: Elder scrolls online intended interrupt behavior.

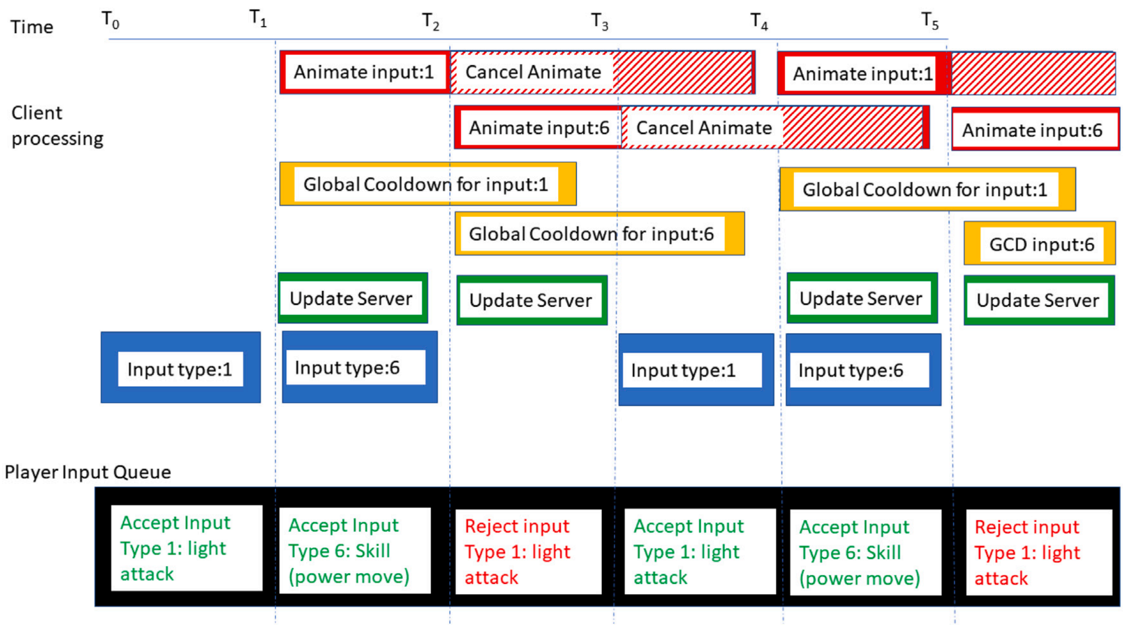

Time
T0 T1 T2 T3 T4 T5
Client processing
Animate input:1
Cancel Animate
Animate input:1
Animate input:6
Cancel Animate
Animate input:6
Global Cooldown for input:1
Global Cooldown for input:1
Global Cooldown for input:6
GCD input:6
Update Server
Update Server
Update Server
Update Server
Input type:1
Input type:6
Input type:1
Input type:6
Player Input Queue
Accept Input Type 1: light attack
Accept Input Type 6: Skill (power move)
Reject input Type 1: light attack
Accept Input Type 1: light attack
Accept Input Type 6: Skill (power move)
Reject input Type 1: light attack

Fig. 11. Scenario 3: Elder scrolls online unintended interrupt behavior.

environmental stimuli within the game. In these situations, the developer chose to allow for interrupts to override the original input processing paradigm, which previously entailed strict delays of user input. Interrupts occur when the user enters an input command of a different type than the currently processed command. The most frequent example of this behavior is when a player initiates an attack against another player, but then chooses to initiate a block command to defend against incoming damage. Fig. 10 represents this behavior, wherein a player initially submitted input for a "light attack" command at time T_0, followed by a "block" command at time T_1. The block command is associated with a higher priority than the previous "light attack" command being processed, and results in an additional input being processed.

Processing the block command initiates an additional update being sent to the game server and requires a new animation sequence to be rendered, thus canceling the in-progress animation sequence of the previous command. Note, even though the block command effectively canceled the animation associated with the previous "light attack" input, there may still exit a global cooldown timer prohibiting additional user input of this type, as is depicted by the user input being rejected at time T_2.

It is important to note that the actual global cooldown timer for an action may be shorter than the animation sequence rendered by the

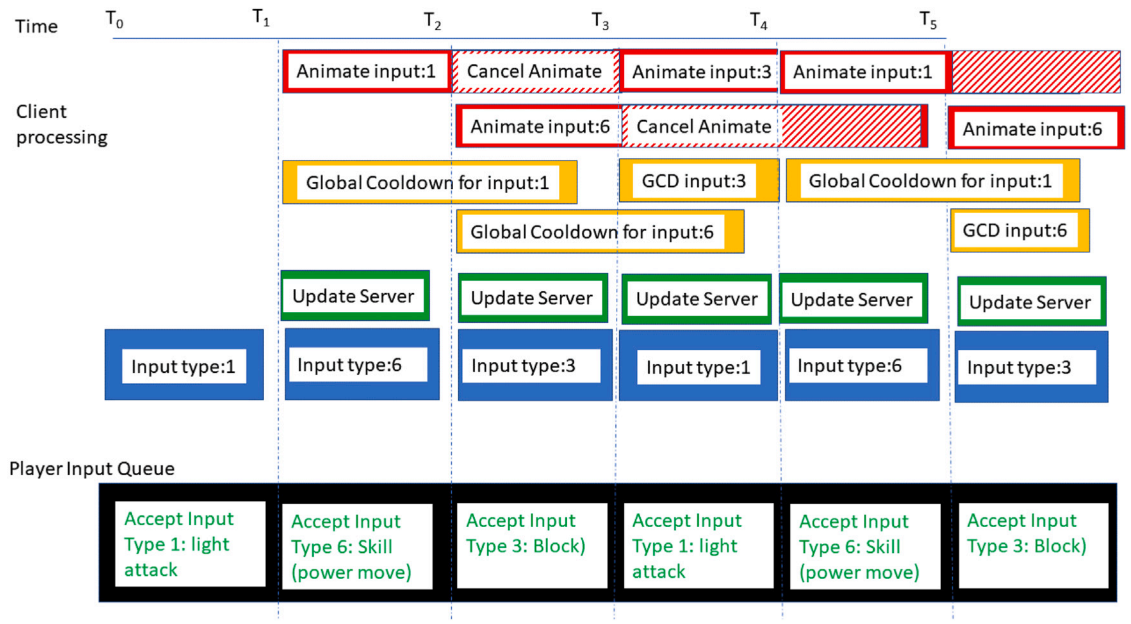

Time
T_0
T_1
T_2
T_3
T_4
T_5
Client processing
Animate input:1
Cancel Animate
Animate input:3
Animate input:1
Animate input:6
Cancel Animate
Animate input:6
Global Cooldown for input:1
GCD input:3
Global Cooldown for input:1
Global Cooldown for input:6
GCD input:6
Update Server
Input type:1
Input type:6
Input type:3
Input type:1
Input type:6
Input type:3
Player Input Queue
Accept Input Type 1: light attack
Accept Input Type 6: Skill (power move)
Accept Input Type 3: Block)
Accept Input Type 1: light attack
Accept Input Type 6: Skill (power move)
Accept Input Type 3: Block)

Fig. 12. Scenario 4: Elder scrolls online unintended throttling behavior.

client. This is because some moves of the same type may have different animations to add variety to the game experience. These variations in animation time may inhibit player actions as they wait for animations to finish prior to processing the next input, as depicted by the model described in Figs. 8 and 9. However, note that at time T_3 in scenario 2, described by Fig. 10, the block command effectively canceled the animation of the previous light attack, and its own animation concluded prior to the beginning of time T_3. This means that the user may effectively enter a new "light attack" command at time T_3 that will be processed by the game client. The introduction of a second "light attack" command at time T_3 indicates the conclusion of a two "light attack" sequence in one less time period than the previously depicted model; therefore, implementing animation canceling in this manner resulted in a 20% increase in light attack frequency.

The ability for a player to sneak in an additional attack in one out of five input cycles by injecting the occasional block command might not have been what the developer of The Elder Scrolls Online intended; however, it also will not necessarily make one player drastically outperform another. That is, unless there was a way to sneak even more actions into the input cycle. Fortunately, for highly competitive players, there is. Fig. 11 depicts the behavior that the game developer admitted was unintended and that is lauded by players as being vital to outperforming their competitors. The key difference depicted in Fig. 11, and what shall be referred to as 'scenario 3', is the introduction of a different type of attack, referred to as a "skill" command, as an interrupt, rather than a block command used in the 'scenario 2' depicted in Fig. 10. "Skill" commands may be used to implement various effects in the game but are generally reserved for the highest damage scoring actions. Therefore, implementing the highest number of "skill" commands within the shortest period will result in the best game performance.

Scenario 3, depicted in Fig. 11, is initiated with a "light attack" command at time T_0, like scenarios 1 and 2 depicted in Figs. 9 and 10 respectively. However, a "skill" command is introduced at time T_1, which now cancels the animation of the command entered at time T_0, issues a new server update to initiate the skill effect, and initiates a global cooldown for future "skill" commands. Note, at time T_2, attempts to introduce another "light attack" command will be rejected due to the global cooldown initiated at time T_0; however, a "light attack" command will be accepted at time T_3 and effectively cancel the animation of the "skill" command introduced at time T_1. This order of commands introduces two "light attack" and two "skill" commands in the same amount of time that just two "light attack" commands would have been processed under 'scenario 1', the intended throttling model. Scenario 3, described in this paragraph, represents the most common application of animation canceling implemented by the Elder Scrolls Online community.

There is however at least one more scenario, as is depicted in Fig. 12. This is the scenario where player input is processed during every possible input cycle and is most likely to occur during player-versus-player (PVP) content. "Light attack" and "skill" command weaving, depicted in Fig. 11, encompass the most efficient input pattern to produce high damage-per-second (DPS) scores, and therefore is the most heavily used pattern in player-versus-environment (PVE) content. However, defensive commands, such as "blocking", "bashing", and "dodge rolling" may also be employed to counter environmental obstacles or other player attacks. Like the pattern described in the previous paragraph associated with Fig. 11, "light attack" and "skill" commands are interwoven to optimize damage output. However, rather than omit a "light attack" command, or issue a rejected "light attack" command, the player may introduce a defensive maneuver, such as "block" during times T_2 and T_5 in Fig. 12. This final scenario represents a near constant flood of user input messages being processed by the game client and sent forward to the game server for additional processing. Unfortunately, the Elder Scrolls Online does not appear to have been designed for dealing with the drastic spikes in network traffic that may appear as a result of this fourth scenario, resulting in drastic variations in both the number of frames per second rendered on the game client and the average response time between client and server traffic from increased network load. Fluctuations in frames per second or network responsiveness both heavily impact the quality of the user experience within the game.

5.3.4. Improving player performance through animation canceling

The Elder Scrolls Online community represents all walks of life. Players are often helpful to one another within the official forums, and many have published guides for assisting new players in navigating the nuances of the game. Some players have even gone as far as to develop custom applications that may be integrated within the Elder Scrolls Online game to increase functionality. One such addon is called "Combat Metrics" and may be used by the player community to generate objective measurements of their damage potential within the game [31]. The Combat Metrics addon is used heavily within the community to evaluate a player's performance potential and therefore standing within the community, which may also influence their ability to participate in game content with other players.

Alcast's animation canceling guide makes use of this add on when comparing character performance without using animation canceling and performance while using animation canceling. The method employed by Alcast mirrors the technique described in 'scenario 3' of the preceding section and illustrated in Fig. 11. Alcast claims that employing animation canceling techniques resulted in an damage-per-second (DPS) increase from 70,000 points to 107,000 points, with no other changes to the character design. This means that animation canceling could potentially result in a 50% increase in performance potential for an experienced player.

5.3.5. *Animation canceling impact on network performance*

The Wireshark protocol analyzer software was used to capture network packets during game play of the Elder Scrolls Online and verify the type of network protocols used. Analysis of captured traffic indicates The Elder Scrolls Online makes use of TCP based protocols exclusively. Overreliance on the TCP protocol to handle all client and server communication within the Elder Scrolls Online could be partially responsible for the challenges Zenimax Online Systems is facing in improving the network performance of their game.

Traffic analysis also revealed that The Elder Scrolls Online game is divided into separate "zones", like how a state may segregate its territories into separate counties and cities. Each of these zones appears to send network traffic from a different IP address indicating that they are likely associated with separate servers processing client requests.

Additional Wireshark packet captures were collected to measure variations in network traffic volume due to player actions in the game. Comparisons between player driven input were conducted within a player character "housing" instance, where no other player characters would be sending commands processed by the server. The Wireshark packet capture indicated that the player housing instance was associated with a separate IP address than other zones with higher player counts, adding credibility to the hypothesis that data collected within the housing instance would reflect a controlled environment.

Player actions replicating scenarios 1, 3, and 4 were monitored via the Wireshark packet capture program for approximately 30 s. The 30 s duration was chosen as in game resourcing mechanisms, such as virtual character fatigue, prevent sustained high rates of action beyond 30 s. The intent of these tests was to measure the potential for peak burst activity in the worst-case scenario. A Logitech G600 programmable mouse was used to store macros of input commands associated with each scenario.

Scenario 1 involved a persistent series of "light attack" actions initiated by a persistent loop of: left mouse click, followed by a 50 millisecond delay, followed by a left mouse release, followed by a 50 millisecond delay; resulting in approximately 10 mouse clicks per second sent to the game client. Scenario 2 was not modeled as it represents reactive player behavior which is expected to occur at random times.

Scenario 3 involved a more complex macro pattern, as merely attempting to loop between alternating "light attack" and "skill" commands resulted in inconsistent input blocks submitted by the client. Ideally, a "light attack" input is entered at the beginning of the client input capture phase, followed by a "skill" input command. If that order is reversed, the higher priority "skill" command may actually impede the "light attack", as "light attacks" cannot "cancel" "skills;" therefore, incurring unintended throttling behavior. To address this, additional delays were introduced into the macro logic. The final macro logic for 'scenario 2' is: left mouse click, followed by a 50 millisecond delay, followed by the left mouse release, followed by a 50 millisecond delay, followed by "skill" command, followed by 100 millisecond delay, followed by left mouse click, followed by 50 millisecond delay, followed by left mouse release, followed by 50 millisecond delay. This pattern ensures that a "light attack" and "skill" weave pattern will complete within 350 ms, fitting two weave attempts nicely within the 1 s client input refresh rate, and increasing the likelihood that "light attack" commands, represented by left mouse clicks, will be the first command entered in subsequent input phases. Even though two weave attempts occur within each input cycle, only the first sequence will be processed due to global cooldowns enforced by the client. The macro repeats in 350 millisecond bursts, rather than clean fractions of a second in order to account for the possibility of input drift incurred from either network lag or initiating the macro sequence midway through a client refresh cycle.

Scenario 4 involved the incorporation of a "block" command, which retains the highest priority and will effectively interrupt any other command. The macro logic was a very straight forward input loop with 50 millisecond delays between "light attack", "skill" and "block" commands, since there is no risk of any of the input commands impeding each other.

Table 1 depicts the number of network packets sent and received during player input testing. While a player's character is idle, the game client sends a single packet approximately every four seconds and receives a packet every five seconds. This may be a heartbeat used to determine if a player is still active before ejecting them from the game. Player actions such as walking or "sprinting" generate a consistent input message to the client and result in an average of 4.14 packets sent and 4.14 packets received per second. The constant input stream generated by moving the character may be used as a baseline for the number of packets the game client expects to process.

Scenario 1, wherein a player clicks their mouse as rapidly as possible to generate "light attack" actions, generates an average of 4.2 packets sent and 4.89 packets received per second. This is a negligible 1.4% increase in packets sent over the baseline and an 18.1% increase in packets received. The increase in packets received above the baseline may be due to additional "damage" scores being generated from random effects associated with "light attack" actions.

Scenario 3, wherein the player initiates a "light attack" action rapidly followed by a "skill" action to cancel the animation of the "light attack", generates an average of 4.96 packets sent and 5.66 packets received per second. This is a 19.81% increase in packets sent and 36.71% increase in packets received over the baseline.

Scenario 4, wherein the player initiates a "light attack" action, followed by a "skill" action, followed by a "block" action, each in rapid succession, generates an average of 11.4 packets sent and 10.9 packets received per second. This is a 175.36% increase in packets sent and 163.29% increase in packets received over the baseline. This is also the first scenario where the number of packets sent is less than the number of packets received. The decreased ratio of messages received to sent may be due to the fact that the "block" action did not generate chance bonus actions like the "light attack" and "skill" actions may, resulting in overall fewer update responses required from the game server.

These tests appear to validate the hypothesis that animation canceling increases the amount of network traffic generated between the client and server. Merely clicking the left mouse button in rapid succession did not generate a flood of network messages, indicating that client input validation has been optimized to reduce superfluous player inputs. Additionally, the modest increase in message traffic from scenario 3 activity, associated with the most popular "animation canceling" technique, also appears to indicate the developer has attempted to optimize network response to this behavior. However, the drastic increase in network traffic from scenario 4, which involves integrating defensive interrupts such as "block" actions within the input stream, indicates that the game is susceptible to extreme network broadcast storms when players replicate this behavior. This may explain why

Table 1
Case study client and server message volume.

Action	#packets sent	#packets recvd	Time (s)	avg packets sent/s	avg packets recvd/s	% sent above baseline	% sent above baseline
Idle	8	6	32.11	0.25	0.19		
Walking	144	144	34.79	4.14	4.14		
Sprinting	132	133	32.15	4.11	4.14		
Scenario 1	142	165	33.77	4.2	4.89	1.45	18.12
Scenario 3	171	195	34.48	4.96	5.66	19.81	36.71
Scenario 4	366	350	32.11	11.4	10.9	175.36	163.29

players who prefer player-versus-player (PVP) type activities frequently complain about network performance or lag. It appears Zenimax Online Systems may also be aware of this phenomenon, as a software patch to client released on May 26th 2020 included optimizations to the way "block" actions were processed in an attempt to curtail "block canceling" behavior [26].

5.3.6. Animation canceling impact on gameplay

As alluded to in the previous sections, implementing animation canceling techniques may increase the number of messages that are sent to, and therefore must be processed by, the game server. Analysis of official forum discussions indicates that player concerns pertaining to network performance are reflected within 35%–60% of discussions that deal with competitive PVE or PVP game play. PVP game play, which is expected to have a larger negative impact on network performance due to its use of scenario 4 in the previous section, exhibits a higher percentage of complaints about network performance within forum discussions than PVE.

Discussions pertaining to player-versus-environment (PVE) game play are expected to contain references to the terms "dungeon", or "trial". This type of game play is limited to "instanced" small group environments within the Elder Scrolls Online game with either four or twelve players at a time participating in a logically separated portion of the game, potentially even hosted on a separate server than the "base" game. "Dungeon" instances are limited to four players at a time and are open to players of all skill levels and experience with the game. 40% of discussions pertaining to dungeons on the forums contain at least one reference to complaints of network performance. "Trial" instances are larger than "dungeon" instances and allow for twelve players to participate in more difficult environmental challenges. Trials are limited to only players who have attained high level characters and therefore are expected to be more experienced at the game. "Trials" participants would be expected to be more familiar with animation canceling techniques and therefore generate a larger amount of network traffic. In fact, it is common practice for elite player groups to prohibit players who are not accustomed to performing animation canceling from participating in trials groups. However, surprisingly, discussions pertaining to trials only contain references to negative network performance in 36% of discussions, which is a 7% decrease from references in dungeon discussions. This may indicate that game servers dedicated to trials content are better suited to handle the increased load, or that more experienced players focus less on what they cannot control, e.g. the amount of latency within the network infrastructure.

Discussions pertaining to player-versus-player (PVP) game play are expected to contain references to the terms "battle ground" or "cyrodiil". "Battle grounds", like dungeons, are small instanced environments hosted on a separate server from the base game and limited to twelve players at one time. Unlike "dungeons", players are playing against one another, with little to no interaction with the virtual environment such as procedurally generated monsters or other typical game challenges. Therefore, PVP game play is expected to be more reactive in nature with players attempting to counter stimuli received from other players' inputs, rather than recurring patterns of procedurally generated challenges. This type of behavior is expected to more closely resemble scenario 4 described in the previously, and is suspected to cause the greatest increase in network traffic, and therefore have a greater negative impact on network performance on game play. Not surprisingly, 55% of discussions dedicated to "battle grounds" contained at least one reference to negative network performance. This is a 12% increase over mentions of network performance within "dungeon" discussions and a 19% increase in mentions over "trials" discussions.

The term "cyrodiil" refers to a special player-versus-player (PVP) environment where hundreds of players may compete against one another. The official claim from Zenimax Online Systems regarding the number of players allowed within the "cyrodiil" environment is 1800 players (roughly 600 for each of the three possible teams or alliances in the game) [32]. However, it appears that the number of available simultaneous players has been adjusted over the years to account for complaints of network performance issues. It is currently estimated by the player community to be approximately 200 players per faction, for a total of 600 players at a time [33]. Not surprisingly, 61% of player discussions pertaining to PVP activities on the cyrodiil servers contain references to negative network performance.

5.3.7. Developer response to animation canceling

The official response from Zenimax Online Systems, the developer of the Elder Scrolls Online, has been mixed regarding the use of animation canceling within the game. The original responses reflected within the forums circa 2014 indicate the developer did not feel that animation canceling was a "game breaking" issue and did not need to devote much attention to fixing the behavior. Later, as the game continued to evolve through consecutive patch cycles, it appears that the developer began to embrace animation canceling as a desirable aspect of the game and even incorporated unique bonuses associated with player attainable items that were triggered by frequently alternating between light attacks and skills within the game. However, the developer's most recent attitude toward animation canceling appears to be focused on limiting its impact on network performance.

Zenimax Online Systems implemented patch 5.3.4 on February 20, 2020 and specifically references modifications to the "core mechanics to block canceling" in order to improve network performance [34]. Several prominent streaming personalities reacted to the changes introduced in the February 2020 patch, to include an individual by the name Thogardpvp. Thogardpvp demonstrated the effect of the new changes in one of his YouTube videos in March of 2020 [35]. Within the Thogardpvp video, it appears that Zenimax Online Systems decided to implement server-side validation checks for "block" commands issued by the player, rather than perform client-side validation. This unfortunately results in a round-trip-time (RTT) delay in processing the player input, as the message must be acknowledged by the server and then processed again by the game client. This results in performance like that of TCP based netcode, which was deemed to be unacceptable within Section 2.1 of this paper. This also appears to be an attempt by the developer to remove the possibility of animation canceling akin to scenario four outlined within Section 5.3.3 and Fig. 12 of this paper. Animation canceling that incorporates "light attack weaving" with "skills", associated with scenario three and Fig. 11 of this paper, does not appear to have been affected by the February patch to the game. Despite an increase in forum posts condemning the changes to animation canceling in February of 2020, player counts measured via steamcharts.com indicate a minor drop in players in the month of February, followed by a massive increase in player count in the month of March 2020. The increase in players is most likely due to the release of a new expansion bringing additional content to the game. However, player counts continue to register above the levels prior to the February 2020 patch, even after the novelty of a new expansion is expected to have dwindled.

6. The impact of client-side and timing exploits in network-based videogames

The first two case study titles: “Risk of Rain 2” and “Dead by Daylight", demonstrated the potential exploitative effects of introducing latency between updates to the underlying game engine. “Risk of Rain 2” demonstrated that procedurally generated effects may be combined to produce unintended chain-reactive behavior which may ultimately overwhelm the ability for game clients to render objects or handle the sending and receiving of game update messages. “Dead by Daylight” demonstrated the ability for savvy players to gain competitive advantage over others by directly controlling access of client update messages sent to the authoritative game server. Both of these titles emphasize the importance of processing game client updates in a timely manner, and in the proper order.

Case study III, based on “The Elder Scrolls Online”, effectively combined aspects contained within both of the previous case studies. Client-side exploitation of intended throttling mechanisms allowed players to produce more game update messages than were intended by the game developer. In case study I, bypassing throttling mechanisms resulted in holistic network degradation for all game clients; however, the state synchronization model depicted in case study III, merely interpreted increased update messages into increased network message generation. Understanding of the network capacity limits inherent to the state synchronization model of "The Elder Scrolls Online” indicates that only a portion of updateable objects may be transmitted in a given update cycle; meaning that periods of network congestion may result in delayed delivery of object update messages. This insight draws correlation between case studies II and III wherein the ability to delay update message transit to the authoritative game server may allow for certain players to gain a competitive advantage over others during real-time game play. This ability to impact game play through manipulation of state synchronization update message queues was introduced as a security concern in Section 4.3 of this paper.

Empirical measurement of network traffic data captured during execution of animation canceling scenarios within “The Elder Scrolls Online” indicate the technique may be used to drastically increase network data generation on demand. Furthermore, anecdotal evidence derived from analysis of social media posts pertaining to “The Elder Scrolls Online” forum indicate a direct correlation between complaints of network performance degradation and the execution of game activities that benefit most from the application of animation canceling techniques. These observations provide credibility to the hypothesis that savvy players may be able to gain a competitive advantage by exploiting client-side throttling mechanisms, and likely at the cost of network performance and satisfaction of other players.

6.1. The role of animation canceling in state saturation

The “Risk of Rain 2” case study, described in Section 5.1, demonstrated the potential for catastrophic performance degradation from the generation of more game objects than can effectively be processed by a game client. The most likely culprit of this performance degradation was the generation of visual objects within game clients; however, it is feasible to consider that network message generation may also increase as a product of responding to similar conditions in network-based games that allow for more than four players, such as an MMORPG.

The “Dead by Daylight” case study, described in Section 5.2, demonstrated the potential strategic advantages of denying state update messages between game clients and servers. This phenomenon was directly measurable based on the ability for players to rotate between client and server roles. Larger games, with dedicated server infrastructure, such as “The Elder Scrolls Online”, do not exhibit this unique peer-to-peer hybrid model; however, asymmetric message generation between game clients and servers is common, and well documented within reference material, such as the HeroEngine documentation.

The network model used within “The Elder Scrolls Online”, outlined in Section 5.3.1, may explain the correlation between increased update message generation and complaints of degraded gameplay due to latency. Network message handling described within the HeroEngine reference material [24] indicates that bandwidth and message count constraints may be applied to decrease the amount of traffic that a game client may generate. Network messages generated by game clients are directly related to player input, but also the product of updating status changes resulting from other environmental effects or actions initiated by other players, similar to the “Risk of Rain 2” case study. Additionally, limitations on update message generation are performed asynchronously, meaning client message generation is throttled, while server message generation is not, similar to the “Dead by Daylight” case study. In the chance occurrence that players generate more update message requirements than are allowed via bandwidth throttling, message prioritization procedures are implemented to send only priority messages during periodic updates. This effectively institutes head-of-line blocking for player input commands, which will have a lower priority than other status updates. Additionally, this explains why social media data pertaining to “The Elder Scrolls Online” seems to indicate increased latency in processing player commands within environments with higher player counts and higher proclivity to animation canceling. Animation canceling techniques, by design, introduce more player input commands into update messages, which will increase as a product of player count. Ultimately, this combination results in eventual throttling of player input messages, as was theorized in Section 4.3 of this paper.

6.2. Recommended solutions to improve game performance

As outlined within Section 2.1 of this paper, a seemingly obvious solution to addressing latency concerns in videogames would be to transition from TCP based network protocols to UDP based implementations. This is often the case in real-time sensitive competitive games, such as first-person-shooter style games. Implementing UDP would decrease the per-packet bandwidth costs of messages, increasing the potential number of messages sent. Additionally, the ability for UDP messages to be discarded, without retransmission, may directly address head-of-line blocking issues with large TCP-based traffic queues.

However, MMORPG games tend to implement TCP based networking almost exclusively. At least eight MMORPG game titles have been identified that allow for animation canceling: “Guild Wars 2”, “The Elder Scrolls Online”, “Everquest 2”, “Lineage 2”, “Aion”, “ArcheAge” and “Star Wars The Old Republic” [36]. All of these titles implement TCP-based networking, with the exception of “Everquest 2” that implements both TCP and UDP-based networking. MMORPG game titles are highly dependent upon data integrity and reliability as their userbase expects to maintain persistence in between gaming sessions. Additionally, the ability to progressively improve or update players’ virtual persona is deeply ingrained within the game experience, and is expected to accurately reflect several months, if not years, worth of gaming time invested into character development. These requirements are often translated into the implementation of a database server, which most likely requires TCP or TCP-like reliability.

Previous research has been conducted into developing alternative protocols for MMORPG titles [37]. This work focused on the performance of theoretical protocols created within the ns-2 network simulator, and ultimately concluded that not all traffic generated within an MMORPG should be treated equally, and therefore exhibits different performance requirements. Drastic performance gains were achieved by segregating traffic into different classes and only implementing costly TCP-like mechanisms on data with high integrity or reliability requirements, such as database updates.

Unfortunately, MMORPG games like “The Elder Scrolls Online” or “Star Wars the Old Republic”, which leverage variations of the HeroEngine network model, appear to encapsulate their update data into a single message format. Bandwidth control in this context, is implemented through head-of-line blocking on the client and reduces the amount of information that will be loaded into the payload of update messages. Modifying the engine to use variable types of replication messages could allow for differentiating between player input actions and other state updates. This could potentially decouple player input actions from being negatively impacted from the effects of state saturation incurred through an excessive amount of other priority update data. Additionally, player input actions could be transmitted via UDP to capitalize on lower latency and prevent network-based head-of-line blocking, while permanent state updates to character status could be reserved for TCP transmission.

7. Conclusions

This paper provided a brief overview of historical and contemporary approaches toward the implementation of networked multiplayer videogames. Three general approaches toward networked game development were presented: deterministic lockstep, snapshot interpolation, and state synchronization. Security concerns associated with potential player cheating were discussed within the context of these three general networking approaches. It is apparent that each game’s exposure to security issues is dependent upon the chosen approach to implementing networking.

Three videogame titles, “Risk of Rain 2”, “Dead by Daylight” and “The Elder Scrolls Online”, were evaluated as case studies depicting exploitable aspects of current networked games. These exploitable aspects were then used to explain the potential impact of “animation canceling” on networked games, specifically the MMORPG genre, and the possibility of state saturation attacks in state synchronization-based network models.

“Animation canceling” appears to be a common aspect of competitive gameplay and is frequently used as a discriminator in determining varying levels of player skill in a game. As such, it has been adopted as a desirable feature in competitive games, though not without potential negative impacts on network and game performance. The case study involving “The Elder Scrolls Online” depicted a potential increase in game client generated network traffic by more than 175% when applying certain animation canceling techniques. Anecdotal evidence from social media data indicates increased references to complaints about network performance in types of gameplay that involve both higher concurrent player counts and proclivity for animation canceling via performance checks (e.g. maximal damage-per-second throughput). The most extreme animation canceling scenario, involving chaotic and reactive gameplay, was attributed to player-versus-player environments within the MMORPG realm. Likewise, social media depicted the highest percentage of references to complaints about degraded network performance within player-versus-player environments.

TCP-based networking protocols may be a contributing factor in degraded network performance for competitive real-time networked videogames. However, reliable data transfer protocols are still necessary within certain game genres, such as MMORPG games, which depend on persistent and consistent long term data storage and retrieval through database systems. Videogame protocols may be partially converted to use a combination of UDP and TCP or TCP-like protocols to address data reliability requirements and apply varying degrees of service guarantees to different types of traffic. This is the approach recommend by this paper to address performance shortcomings, while maintaining the possibility of player skill differentiation via leveraging animation canceling techniques.

CRediT authorship contribution statement

Blake Bryant: Conceptualization, Formal analysis, Writing - original draft, Data curation. **Hossein Saiedian:** Conceptualization, Methodology, Validation, Resources, Writing - review and editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

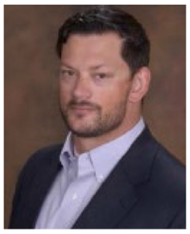
Acknowledgments

The authors would like to thank the following students from the University of Kansas IT780 (Communication Networks) course for performing experimentation in support of the “Dead before Daylight” case study: Niels Hansen, Annika Kuhnke, Aravind Pothuri, and Nayab Shaik. The authors would also like to thank Logan Bryant and Gabriel Bryant for their support in demonstrating the “Risk of Rain 2” exploit referenced as case study I in this paper. Finally, the authors would like to thank the anonymous reviewers for their helpful comments and work toward improving this paper.

References

[1] K. Webb, Business insider, 2019, https://bit.ly/36SDCAw, Retrieved from The Fortnite World Cup Finals start this Friday, and $30 million is on the line. Here’s what you need to know about the competition: businessinsider.com.
[2] D. Hanson, The 10 richest twitch streamers in 2019: moneyinc.com, 2019, https://moneyinc.com/richest-twitch-streamers-in-2019/.
[3] J. Carmack, Quakeworld by John Carmack. Retrieved from fabiensanglard.net, 1996, http://fabiensanglard.net/quakeSource/johnc-log.aug.htm.
[4] Y. Gu, R.L. Grossman, UDT: UDP-based data transfer for high-speed, Comput. Netw. 51 (2007) 1777–1799.
[5] B. Eckart, X. He, Q. Wu, Performance adaptive UDP for high-speed bulk data transfer, in: 2008 IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–10.
[6] Z. Yue, Y. Ren, J. Li, Performance evaluation of UDP-based high-speed transport protocols, in: 2011 IEEE 2nd International Conference on Software Engineering and Service Science, IEEE, Beijing, China, 2011, pp. 1–5.
[7] Y.-T. Li, D. Leith, R.N. Shorten, Experimental evaluation of TCP protocols for high-speed networks, IEEE/ACM Trans. Netw. 15 (5) (2007) 1109–1122.
[8] S. Biplab, S. Kalyanaraman, K.S. Vastola, Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno, and SACK, IEEE/ACM Trans. Netw. 11 (6) (2003) 959–971.
[9] G. Fiedler, Physics for game programmers : networking for physics programmers, in: Game Developer’s Conference, (pp. Video Recording https://bit.ly/31z4nbj), San Francisco, 2015.
[10] Inc. Epic Games, Unreal networking architecture, 2012, Retrieved from UDK Networking Overview https://bit.ly/30BZ9Ml.
[11] J. Van Waveren, The doom III network architecture, 2006, Retrieved from mrelusive.com: https://bit.ly/304GBVv.
[12] A. Dainotti, A. Pescape, G. Ventre, A packet-level traffic model of starcraft, in: Second International Workshop on Hot Topics in Peer-To-Peer Systems, IEEE, San Diego, 2005.
[13] C.-S. Lee, The revolution of starcraft network traffic, in: NetGames ’12 Proceedings of the 11th Annual Workshop on Network and Systems Support for Games, IEEE, Venice, 2012.
[14] Valve, Networking entities, 2019, Retrieved from Valve Developer Community: https://developer.valvesoftware.com/wiki/Networking_Entities.
[15] Valve, Source multiplayer networking, 2019a, Retrieved from Valve Developer Community: https://bit.ly/3ks3UQS.
[16] M. Claypool, D. LaPoint, J. Winslow, Network analysis of counter-strike and starcraft, in: Conference Proceedings of the 2003 IEEE International Performance, Computing, and Communications Conference, IEEE, Phoenix, 2003.
[17] M. Frohnmayer, T. Gift, The TRIBES engine networking model, in: Proceedings of the Game Developers Conference, 2000, Retrieved from gamedevs.org: https://www.gamedevs.org/uploads/tribes-networking-model.pdf.
[18] Valve, Valve anti-cheat system (VAC), 2017, Retrieved from Steam Support: https://support.steampowered.com/kb/7849-RADZ-6869/#whatisvac.
[19] Hopoo, Hopoo games development thoughts #17, 2020, Retrieved from Steampowered.com: https://bit.ly/3d612ao.

[20] Astephen542, Forgive me please soulbound catalyst = OP. Retrieved from reddit, 2020, https://bit.ly/3t92df5.
[21] SeeDee, What is lag switching? 2017, (Online forum post). Retrieved from Steam Community: https://bit.ly/3d9Akhc.
[22] C. Talbot, Elder Scrolls Online player count hits 15 million: Pcgamesn.com, 2020, https://www.pcgamesn.com/the-elder-scrolls-online/player-count.
[23] A. Biessener, Why the elder scrolls online isn't using heroengine, 2012, Retrieved from Gameinformer.com https://bit.ly/3wHeM3r.
[24] HeroEngine, Heroengine wiki. Retrieved from hewikiheroengine.com, 2012, http://hewiki.heroengine.com/wiki/Replication_Tutorial.
[25] Steamcharts, Elder scrolls online: Steamcharts.com, 2020, https://steamcharts.com/app/306130.
[26] The Elder Scrolls Online Forum, PC/Mac patch notes v6.0.5 - Greymoor & update 26: The elder scrolls online forum, 2020, Retrieved from : https://beth.games/3a7BWF9.
[27] Uberkull, Animation canceling good for the game? 2015, Retrieved from The Elder Scrolls Online Forum: https://beth.games/30FDeEb.
[28] Deltia, ESO animation canceling guide, 2014, Retrieved from Deltias gaming http://deltiasgaming.com/2014/10/16/eso-animation-canceling-guide/.
[29] Naranarra, Weaving beginner guide animation canceling for elder scrolls online. Retrieved from altcasthq, 2019, https://alcasthq.com/eso-weaving-beginner-guide-animation-canceling/.
[30] Arzyel, ESO- how to increase your DPS | animation canceling guide, 2019, Retrieved from YouTube Channel https://bit.ly/2XCAwgK.
[31] Decay2, Combat metrics, 2020, Retrieved from https://bit.ly/33Bw1qu.
[32] Elder Scrolls Online Forum, How will the player population be limited? 2015, Retrieved from The Elder Scrolls Online Forum: https://help.elderscrollsonline.com/app/answers/detail/a_id/6533.
[33] Elder Scrolls Online Forum, How many players are allowed in a campaign? 2016, Retrieved from The Elder Scrolls Online Forum: https://beth.games/2XEbL3H.
[34] Elder Scrolls Online Forum, PC/Mac patch notes v5.3.4 - harrowstorm & update 25, 2020, Retrieved from The Elder Scrolls Online Forum: https://beth.games/3fE31AP.
[35] Thogardpvp, The problem with removing animation cancelling, 2020, Retrieved from Youtube: https://www.youtube.com/watch?v=CIzNq2exFHs.
[36] MMORPG Forum, Discussion / addressing animation canceling in games, 2015, Retrieved from MMORPG.com: https://forums.mmorpg.com/discussion/432344/addressing-animation-canceling-in-games.
[37] C. Wu, K. Chen, C. Chen, P. Huang, C. Lei, On the challenge and design of transport protocols for MMORPGs, Multimedia Tools Appl. 45 (1–3) (2009) 7–32.



Blake D. Bryant, MS, CISSP, MCITP, CCNA, completed his MS degree in information technology at the University of Kansas in spring of 2016 and is currently pursuing his Ph.D. degree in Computer Science. He is a security professional with 15+ of experience in leading computer security organizations (private and military) and is currently serving as a professor of practice teaching courses in information technology at the University of Kansas

Hossein Saiedian (Ph.D., IEEE PSEM, Kansas State University, 1989) is currently an associate chair, the director of IT degree programs, and a professor of computing and information technology at the Department of Electrical Engineering and Computer Science at the University of Kansas (KU) and a member of the KU Information and Telecommunication Technology Center (ITTC). Professor Saiedian has over 160 publications in a variety of topics in software engineering, computer science, information security, and information technology. His research in the past has been supported by the NSF as well as other national and regional foundations.