Networking in Early Video Games

Benjamin Schmidt
University of Calgary
Calgary, AB, Canada

John Aycock
Supervisor
University of Calgary
Calgary, AB, Canada

ABSTRACT

We discuss five of the earliest networked video games – *Maze* (1973), *Sopwith* (1984), *SGI Dogfight* (1985), *Habitat* (1986), and *Netrek* – each with distinctly different networking architectures. After analysis of the source code, we break down implementation details and compare similarities and differences in their approaches to real-time communication between players. We also use first-hand evidence backed up by implementation details to shine light on flaws in the systems.

CCS CONCEPTS

• **Social and professional topics** → **History of software**.

KEYWORDS

Networked video games, Networking, History of video games

ACM Reference Format:
Benjamin Schmidt and John Aycock. 2024. Networking in Early Video Games. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Since the very early days of video game history, players have competed or collaborated with each other in multi-player game modes [Brookhaven National Laboratory 2013]. This was achieved by connecting multiple separate input devices or a single shared device to a single computer, and proved effective for players near enough to the computer but inadequate for any distance beyond the length of an input device's cable. It was also infeasible for more than a handful of players to play together. Modern networked video games have solved these problems, allowing hundreds to millions of players to play together simultaneously from anywhere with an internet connection. We will discuss the early steps of the journey to achieve this by looking at three of the earliest networked video games – with source code available – from the 1970s and 1980s.

These early video games are *Maze*, *Sopwith*, and *SGI Dogfight*. *Maze* and *Sopwith* were created before the ubiquity of the TCP/IP stack, and *SGI Dogfight* in the very early days of TCP/IP. We will discuss their solutions for networking without reliance on the standard protocols we know and rely on today. This will be achieved through static analysis of the source code and reports from the original developers or players. We aim to emphasize the significance of each game, its unique solution to networking, and any issues that it may have faced.

2 RELATED WORK

Much of the discovered work in the field of networked video games describes systems with broad application that can be implemented in a number of ways. For example, Das et al. [1997] describe a general architecture – NetEffect – with which multi-user video games or virtual worlds can be created, giving the example of a particular implementation built to demo NetEffect. The authors describe a number of interesting ideas, such as distributing the virtual world over multiple servers which handle subsets of that world, peer-to-peer voice chat, load balancing, and a number of other techniques primarily intended to reduce network traffic, especially inter-server traffic. However, it doesn't appear that NetEffect was used widely, and its emphasis on scalability was not something we found in the earlier networked video games that this paper analyzes. Furthermore, we will discuss concepts specific to the various games studied and perform a comparative analysis on their implementation, effectiveness, and flaws from a contemporary perspective.

Blau et al. [1992] follow a similar pattern. They describe Virtual Environment Realtime Network (VERN) – an extension of DARPA's SIMNET which enabled physically distant users to interact and communicate in a virtual environment. VERN, like NetEffect, is a framework in which to build virtual worlds, and so this paper is once again different to what we present.

In other literature, the focus is on more specific solutions rather than generalized frameworks. Singhal and Cheriton [1995] describe an alternative approach for positional information transmission – instead of sending real-time position data or using dead-reckoning, position data with a timestamp is periodically sent instead, and clients estimate the object's position, velocity, and acceleration accordingly. They claim this solves an problem exhibited by *SGI Dogfight* where the frequency of packet transmissions (tied to frame rate) affects smoothness of the rendering of other players. It would also mitigate the network saturation caused by the game's traffic when enough players were present.

Bernier [2003], a Software Engineer at Valve Corporation (then Valve Software) presents solutions for latency compensation in the context of an architecture with a single authoritative server and many "dumb" clients. In essence, he describes how a user can feel as if they are interacting with other clients in real-time as they would in a peer-to-peer architecture, having the security benefits of a server-orchestrated architecture – clients cannot broadcast their own state as an invalid one, for instance teleporting their character or changing the state of their score – while mitigating the downsides involved with high latency or connections.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Das et al. [1997] and Blau et al. [1992] both note the use of dead-reckoning, in which clients only broadcast changes in direction or speed of a controllable game object, and are otherwise assumed to be moving at the same speed and in the same direction as they were at the time of the last broadcast [Blau et al. 1992]. This is in contrast to earlier video games which instead updated the character's current location and direction every time it changed, which naturally used significantly more bandwidth than dead-reckoning.

Overall, the body of literature covers generalized topics – either broad systems or specific implementation details which may avail developers when building a networked game.

Other related works discuss the history of video games, in general or for specific games. For example, the IEEE Annals of Computing History Volume 31, Issue 3 is dedicated to the history of video games and contains articles detailing the history of *Pong* (1972) [Lowood 2009] and *Nimbi* (1963) [Jorgensen 2009], and how various display technologies were used in video games [Bogost and Montfort 2009] among others.

Our work is adjacent to the cross-disciplinary field of archaeogaming, where old video games are treated as archaeological artifacts to be preserved, restored, and studied using archaeological techniques within the context of their time and the people who played them [Aycock and Biittner 2023]. We lack the archaeological connection, but hope to accurately portray the technical details of the networked video games we discuss.

Works in this field consist of uncovering implementation details of retro games as well as the human aspect behind their creation and enjoyment. [1] For instance Aycock and Copplestone [2018] reverse engineer the central maze-generation algorithm behind *Entombed* (1982) for the Atari 2600. By discovering a bug in this code, they were able to show reuse of this code in other Atari games, and used this as a vehicle to discuss the programmers behind it.

Perhaps the most important work to our review is *The Lessons of Lucasfilm's Habitat* (2008) [Morningstar and Farmer 2008]. It is authored by the original creators of *Habitat* (1986) as a retrospective on the game, touching on technical challenges faced, implementation and architecture details and decision-making, and emergent behaviour observed from the player-base. This is an excellent primary source which gave us plenty of information when researching *Habitat*. The paper goes on to advise future developers of virtual worlds with the lessons learned as well as ideas for future improvements to be made and the developers' philosophy on virtual worlds as a whole.

3 PRELIMINARY

In this review, we will examine the networking implementations used by a variety of early networked video games, assess their implementation details and speculate on their performance and potential flaws. For the purposes of this research, a networked video game may be defined as a video game in which two or more independent clients running the same code interact over a real-time link. Each client therefore handles I/O, networking, and at least some portion of game logic.

[1] This description does not encompass all the work involved in archaeogaming. See [Aycock and Biittner 2023] for a more detailed explanation of archaeogaming.

Many early video games supported multiplayer functionality, allowing players to compete or co-operate in real time. However, these early solutions were typically multiple input methods connected to a single device with a single display. Although this was effective – and indeed is still used in contemporary games – geographically separated players had no means by which to interact. The PLATO system featured the ability to host multiple graphics terminals (thin clients) which served to simply relay input and output to a mainframe which performed the processing. A number of games were built using it, allowing users to play together even if they were in a different room, and as many mainframes running PLATO were networked together, the terminals were effectively networked together as well [Jones and Latzko-Toth 2017]. Despite the importance of these games, they do not fall under our specifications of a networked video game – the network code is restricted to the PLATO system on the mainframe itself rather than the client.

Within the scope of our definition, there were some common technologies used by the games we will examine. RS-232 serial connections were used, either as a direct connection between two clients (the serial cable functioning as a null modem), or connecting all clients to a central mainframe which could would broadcast data received from one client to all others. According to the specification, these cables were limited to 20kbps transfer speed [Buchanan 2004], although many were used in unintended ways and specified limits and uses weren't always followed. These cables were effective as null modems when connecting two clients, but this is not part of the RS-232 standard, and had speeds lower than alternatives (like Ethernet), making them not suitable for network transfer beyond simple peer-to-peer two-player implementations.

In 1980, Ethernet was published as an open standard by a partnership between Xerox, DEC, and Intel [Digital Equipment Corporation et al. 1980], being standardized by the IEEE in draft form as IEEE 802.3 in 1983 [IEEE 1985]. Early Ethernet used a shared cable where all connected devices would receive the same data, resulting in collision risk and shared bandwidth. Despite its limitations, Ethernet was simpler, cheaper, and more effective than its competitors, and it became the *de facto* standard for wired local communication [Metcalfe and Boggs 2006].

Lastly, Transmission Control Protocol and Internet Protocol (TCP/IP) was made the sole transmission protocol on the ARPANET in January 1983 [Postel 1981], which persisted as ARPANET transitioned into the public internet, and of course is still in use today.

Before discussing the games, we will elaborate on the methodology used to create the final list. Wikipedia was chosen as the source for finding games due to the amount of data available and well-organized categorization. A Python program (Source Code available on GitHub) was written by me, making use of the `pywikibot` library [MediaWiki 2024] to pull the `.wikitext` file for every article in the categories "Category:<year> video games" for every year between 1970 and 1989 (inclusive) i.e., the article for every video game published between 1970 and 1989. Once this list of nearly four thousand games was extracted, keywords were searched for within the article text by the program searching for at least one reference to a keyword in the text. More complex means of identification were attempted but were found to reject a large number of valid candidates, and so the aforementioned method was used in conjunction with a manual review of the final short-list of just

under two hundred video games. This manual review consisted of discarding articles in which the only references to the keywords were unrelated to the game (e.g., citations of publications with "network" in their name). Sixteen games were left, and of these, five were chosen for their available source and significance. The three games discussed here are the first three in chronological order by release year.

4 FINDINGS

4.1 Maze

Maze (or *Maze War*) is the oldest of the video games we discuss in this review. First created in 1973 by high school students working at NASA in the summer of the same year [Thompson 2004], it is the first networked video game by our definition. [2] Players would simultaneously occupy a shared maze, and had the ability to move forward, backwards, left, and right, rotate in 90° increments, and fire missiles in the direction they faced. The objective was to shoot other players, which would increase the shooter's score if successful, with scores being displayed in an in-game leaderboard. *Maze* also supported a few additional features including text chat.

Maze was written for the Imlac PDS-1 in its native assembly language, and initially supported multiplayer functionality by connecting two PDS-1s with a RS-232 serial cable [Thompson 2004]. In 1974, development continued at MIT, where multiple PDS-1s were connected to a central DEC PDP-10 server. This version of *Maze* is what we will discuss henceforth, as the original source code is fully available and has been open sourced.

We will briefly discuss the server's function. The server could support up to eight PDS-1s, each connected to it by a separate serial cable, and enabled initial setup and message "routing" [Thompson 2004]. Upon connection, the game and a custom maze (if specified) were downloaded from the server to the client, and players were connected together. From this point on, the server acted as a repeater of characters from a PDS-1, forwarding them to all connected devices except the sender. All game logic and network code is therefore handled by the client, with the exception of "robots" – characters controlled by the server instead of a human player. As PDS-1s were common in institutions that formed the ARPANET, cross-institutional play was possible for mainframes connected to it, and so the server would handle this as well.

Maze used the *Maze* Protocol to send and receive information about player and game state. Each client would maintain a copy of the global state locally. Information about other players was updated when receiving messages as well as when the local client shoots another player. Clients would simply receive characters from the serial connection by calling an instruction to move the input from the buffer to a register, masking to the lower 7 bits – discarding the most significant bit to fit the ASCII standard – then handling that character as the input. This would happen in an infinite loop, no different than standard practice for modern real-time networking. A single character was not enough for useful information, so multiple characters would be sent in this structure: `<command> <arg1> <arg2>. . . <argn>`. Each character was sent separately, and the receivers would wait for all the arguments to arrive (if needed) before processing the command. With this in mind, it would theoretically be possible for characters from multiple clients to be interleaved, with bytes from one player being mixed in with those from another, causing unpredictable behaviour. It is unclear if the server had a role in mitigating this, but we do know that *Maze* had error handling for unexpected data. For example, player IDs are checked for validity (between `0o001` and `0o008` inclusive) and that they are not the receiving player's ID. There are also checks for turning in an invalid direction, or moving to a location outside the maze.

The following is a description of the six possible legal message types. In the description below, we separate each character by whitespace. No arguments are optional. [Palmer et al. 1974]

> `0o001 <id>`
> Player `id` left the game.
>
> `0o002 <id> <direction> <x> <y>`
> Player `id` moved to position `(x,y)` and turned to `direction`.
>
> `0o003 <id> <target id>`
> Player `id` shot player `target id`.
>
> `0o004 <id> <uname> <# of hits> <# of deaths>`
> Player `<id>` with name `uname` joined the game.
>
> `0o014`
> Erase display ring buffer, clears message history.
>
> `>0o014`
> Print this character in the display ring buffer. For example, if `0o015` is received, a carriage return is printed.

These are mostly self-explanatory, but some further detail is needed for message types `0o002` and `0o004`. When a number is sent (e.g., `# of hits`), the `0o100` bit is set to distinguish this from an ASCII character. If characters are received when numbers are expected, the message is discarded by error handling.

A player's `uname` is always 6 characters long with spaces used to pad shorter names, and the count of hits and deaths are sent as two characters each, the first character carrying the high order 6 bits, and the second the lower order 6 bits. These would be combined to form the full 12-bit number. As a result of these, message type `0o004` would send 12 bytes total.

Message type `0o004` is sent when a new player joins to announce their presence and username. In response, all clients would reply with a message of the same type so the new player would know all other players' usernames and scores.

As mentioned earlier, the PDS-1 was common in the ARPANET, and according to Thompson [2004], half of the traffic between Stanford and MIT were allegedly from *Maze* games in progress. The considerable distance between these two institutions (~ 4000km) created considerable latency which affected gameplay, though this was fixed in a later version by sending data about the relative motion in type `0o002` messages.

Latency was an issue due to the client-side hit registration. For instance, player 1 knows it has shot player 2 based on the latest location it has of player 2. However, if player 2 moved out of the line of fire before receiving the full message informing them of their death, from their perspective it would appear as if they were

[2] *Spasim* (1973) was released in the same year for the PLATO system with support for up to 32 simultaneously players, though exact release dates are unclear.

hit by an enemy who shouldn't have been able to see them. This would give an unfair advantage to the shooter when latency was sufficiently high.

Other issues may have also included the aforementioned message interleaving, though many cases would likely have been handled by the error handling in the client code.

4.2 Sopwith

Released in 1984 by Canadian company BMB Compuscience as a demo for Imaginet – their proprietary networking system – *Sopwith* is a simple video game where up to four players fly their own Sopwith Camel aircraft from a side-on 2D perspective. A more modern version of the code written by David L. Clark is available and open source, but the original networking components are still present, which we will look at.

Sopwith itself does not implement much networking, and heavily relies on the Imaginet protocol, so we will discuss this first. Imaginet was a hardware adapter built by BMB Compuscience which could connect Atari STs, IBM PCs, and other popular computers of the time [Cole 1986]. MS-DOS introduced native networking later in 1984 with DOS 3.1, but DOS 2.1 was still sold in alongside it [Necasek 2011], and IEEE 802.3 was still in its infancy, so Imaginet found a niche in the market. Imaginet was essentially a single floppy drive shared by all connected clients. The host would either have a physical drive or a virtual floppy drive image on a hard disk drive. Clients would have hardware installed in the place of a floppy disk controller, making Imaginet fully transparent to clients, appearing to the computer as a normal floppy drive. This meant it could be compatible with any device that supported floppy drives, regardless of operating system, and any software could become networked by addressing the shared disk. To prevent race conditions when multiple devices try to access the same file simultaneously, a microprocessor in the hardware controls access to the disk, marking it as `BUSY` while in use [Agnew et al. 1983].

As for *Sopwith*, it makes use of Imaginet using a custom interface called Diskette I/O or DKIO. The implementation is very simple – if the game is in multiplayer mode, the multiplayer state is saved to and loaded from the virtual floppy drive. The state includes all relevant multiplayer information like player count, max players, state, physics, fuel, score, and physics information of each player. The state in the local memory and the state on the server is synced once per game loop, i.e., at frame-rate. An unusual implementation detail is that all clients write to the same specific virtual "disk sector" rather than using some kind of file system lookup, an interesting example of the transparency of Imaginet. This would prove problematic if the sector stored other data, overwriting or corrupting it, but since *Sopwith* was intended as a demo, this was likely not a concern.

In more modern times, Imaginet has been emulated by a project called Imaginot [Howard 2024]. Imaginot works by intercepting and responding to low level DOS system calls, sending the inputs over a specified network protocol, allowing *Sopwith* to be played in multiplayer mode without needing the original Imaginet hardware. Although this is beyond the scope of this project, it is interesting since Imaginot is transparent to Imaginet, which itself is transparent to *Sopwith*, and so running *Sopwith* now requires two layers of networking technology which act as virtual devices, a good example of the challenges in early video game networking before TCP/IP became standard.

4.3 SGI Dogfight

After its standardization on the ARPANET in 1983, TCP/IP was first used in a networked video game by *SGI Dogfight*. Originally created as a demo of the graphics capabilities of Silicon Graphics Inc. (SGI) workstations as well as SGI's IRIS GL – their proprietary graphics API which later became the open standard OpenGL – *SGI Dogfight* is a 3D flight simulator which was included with SGI workstations [Zyda 2000]. At first it was limited to single-player gameplay, but multiplayer was added in the form of a direct serial link (similar to how *Maze* networking began), then over Xerox Network System (XNS) multicast over Ethernet. No interaction between players was possible, but other player-controlled planes were visible. In 1985, `dog` was created, which allowed players to engage in dogfights, and UDP networking was added in 1986. This is significant as it marks the first known time the TCP/IP stack was used in a video game. SGI made the source code available on request, and many developers reportedly used the code to teach themselves how to use UDP packets [Zyda 2000]. Source code dated to 1988 is available and open source [Tarolli et al. 1988]. Though some changes may have been made from the original, the networking code appears to be unchanged and exhibits issues noted by developers and players, so this appears to be a representative example of the original UDP code.

Like *Sopwith*, *SGI Dogfight* sent and received multiplayer data once per frame. Communication was achieved by using UDP broadcast packets through port 5130, sending them to every client on the network. If other clients were running *SGI Dogfight*, they would interpret the packets to update their local state. This implementation would prove problematic as we will discuss later.

SGI Dogfight was built for IRIX, SGI's UNIX-based operating system. As such, it uses the `<netinet/in.h>` POSIX header to interface with the kernel-level networking. On initialization, structures are created for each player-controlled plane which stored the same full set of information as the local plane. The broadcast socket and host's IP address are found and stored for subsequent use. Each plane is given an ID (the client's IP address), and lookups are performed by iterating through the plane structs until a plane with the matching IP address was found and returned. Depending on the number of players present, it might have been more efficient to create a mapping of numerical IDs to IP addresses and directly address the corresponding element in the array when looking up a specific plane, though this would trade off a small amount of memory.

Sending and receiving is implemented in a simple way – sending uses the `sendto` function directly, and receiving reads from the socket in a loop until no more data is available in the socket buffer. The packet structure sent is always a plane struct, regardless of message type, so the `cmd` field dictates this message type, represented by a macro corresponding to an integer. The following is a description of each of the 4 packet types. [Tarolli et al. 1988]

```
DATA_PACKET 0
```

> Standard plane struct with all plane-related information.
>
> `MSG_PACKET 1`
> Contains chat message and addressee; if no addressee is specified the message displays for all players.
>
> `SUPERKILL_PACKET 2`
> Special admin message which exits the game for the addressed player or all players if none is specified.
>
> `KILL_PACKET 23`
> Exits game if game version of sender is greater than the receiver's.

Despite using the plane struct, chat messages are just written to the struct from a certain point onward, ignoring the normal structure. Regardless of whether an addressee is specified, the message is still broadcast to all players, and the receiving code decides whether or not to display it.

SGI Dogfight has handling for mismatched versions as alluded to in the packet schema – if a message is received and the sender's game version is strictly less than the receiver's version, the receiver will respond with a `KILL_PACKET` addressed to that sender. This ensures that all players must have the same version as the player with the newest version and so achieves a form of eventual consistency of version.

`DATA_PACKET`s corresponding to the local plane struct are sent every frame to update the locally stored information for all other players. When receiving a `DATA_PACKET`, the receiver also checks if a certain field in the packet is the same as their ID in order to run the function to crash their plane.

This implementation has a few problems. Since source code was available, it would have been trivial to modify the program to be able to send a `DATA_PACKET` to another player which would crash their plane without needing to fire a shot. Malicious users could also send `SUPERKILL_PACKET`s to force others games to exit.

More notably, a game of *SGI Dogfight* with sufficient players acts as an unintentional Distributed Denial of Service (DDoS) attack on other devices attached to the network. This problem was exacerbated by *SGI Dogfight*'s inclusion in all new SGI workstations, making it very easy to access. One `DATA_PACKET` is a struct of size 112 bytes, sent once per frame, and workstations could run *SGI Dogfight* between 7 and 15 frames per second depending on hardware. This means one player broadcasts 784–1680 bytes per second during gameplay. The total traffic on the network is of course multiplied by the number of players, and the volume would cause congestion in a network if enough players were in a session, blocking traffic for other devices and even freezing them outright [Mace 1988]. The game even includes an error message which warns of this:

> WARNING some machines can not handle large numbers of udp broadcast packets. If you have machines from other vendors on your network, running dog on your network may bring them to a halt. VAXes are known to have this problem.
> [Tarolli et al. 1988]

4.4 Habitat

In 1986, *Habitat* was released for the Quantum Link online service for the Commodore 64 home computer. Quantum Link would later evolve into America Online [Nollinger 1995], the first service to bring online access to the mass market.

The Commodore 64 is the most sold single computer model of all time [Guinness World Records 2024], capturing between 30 to 40% of the United States market share [Reimer 2005]. It was because of this popularity that the Commodore 64 was chosen as the platform for the client.

Habitat is a "multi-player online virtual environment" [Morningstar and Farmer 2008] and one of the first large-scale networked video games, supporting thousands of players in one shared world, a significant advancement from the earlier games we discussed. Players control virtual avatars which can interact with both the world around them and other players' avatars. The game provided no concrete objectives or rules but instead served as a medium for communication and interaction, allowing players to live out a virtual life. It is distinct from the bulletin board systems (BBSes) which were common at the time – even on Quantum Link – thanks to its virtual world, graphical representation thereof, and virtual avatars which could be separate from the human at the keyboard. Although BBSes supported messaging and online games [Edwards 2016], there is no evidence that these had any relation to each other unlike *Habitat*, where virtual rewards could be earned through gameplay and were visible to other players if displayed on the avatar.

Habitat was made up of many interconnected "regions" which made up one screen's worth of content, containing objects and players. Players could only interact with the world and other players that were in the same region as them.

Due to its much larger player capacity and the desire for data integrity, *Habitat* used a client-server architecture, with users' Commodore 64 PCs operating as the client and the Quantum Link backend as the server. The backend held the representation of the world and was responsible for communicating this with the clients, handling client messages, updating world state, dealing with connections, and storing persistent data. The system operated on a zero-trust model where the server should not assume that any client data is valid, and so every message must be checked and validated before updating the world state. As such, clients would simply make requests to the server which would respond appropriately with the changes to the world state, and the client would render this. This meant that the client was only responsible for rendering, local memory management, I/O, and network communication. The lead developers strongly believed that an Object-Oriented paradigm was essential for multi-user worlds [Morningstar and Farmer 2008] and so the entire *Habitat* world was made up of Objects. Objects could respond to incoming messages and change state accordingly. If no players were in a region, objects would be written to a persistent database and unloaded from the server's memory.

Communication was facilitated over a dial-up connection via a modem attached to the Commodore 64's RS232 port. *Habitat* required at least a 300bd connection. Traffic was routed through a packet-switching network within the Quantum Link servers, which

were fault-resistant Stratus minicomputers. According to information sent by Quantum Link to Lucasfilm, the expected error rate on packets was 1 per week requiring retransmission and 1 per year escaping detection[Morningstar et al. 1986a]. This routing could introduce between 100 and 5000ms of latency. The inter-server communication is not documented, and is outside of the scope of this review, so we will treat the backend network as a single server which matches how the client code is written.

The source code for *Habitat* is published under the MIT license by The Museum of Art and Digital Entertainment [Morningstar et al. 1986a]. Analysis of multiple versions of the source code, documentation by the developers, notes, thoughts, and email records was done on this repository, which will make up the remainder of this section.

The Commodore 64 source code is written in the Macross 6502 cross assembler, developed by Chip Morningstar for *Habitat* [Morningstar et al. 1986b]. The Stratus source code is written in IBM's PL/1.

Quantum Link provided software for the client to use to interface with the network, but this used one of the four 16KB memory banks of the Commodore 64, reducing the memory available for the client to store local information about the world. This limitation meant that the client software needed to be written from scratch in an interrupt-driven fashion, which is what was used in the final product. Quantum Link did not use a standardized networking protocol like TCP/IP, and had its own unique link layer packet structure, described here:

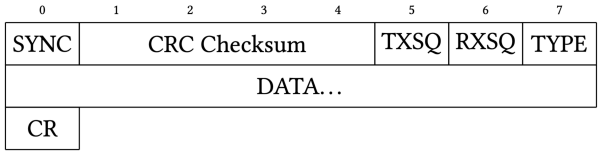

Figure 1: Quantum Link Link Layer Packet Structure. Units in Bytes.

SYNC	delimits start of packet
CRC	4 byte CRC checksum
TXSQ	transmission sequence number
RXSQ	reception sequence number
TYPE	type of data
DATA	payload
CR	delimits end of packet

It appears the Quantum Link developers were concerned about data integrity, dedicating 2 bytes to start and end delimiters, 4 bytes to checksum, and 2 bytes to sequence numbers. This meant that it could detect corruption (using the checksum) or out-of-order transmission or reception (using sequence numbers). In total, a Quantum Link packet has 10 bytes of overhead for packet headers. This high overhead was criticized by Farmer, who suggested improvements to reduce this overhead by removing the SYNC byte and compressing the checksum, although these were not implemented.

DATA packets are considerably more complicated than those we have seen in the previous games. As such we will explore a few scenarios of packet transfer instead of describing all the details. At a high level, packets are directed from the client to a specific object on the server, which may send a message in response. The server may also send packets to the client to inform it of changes to the world state that were not triggered by the client, typically from another player's interaction with the world which is relevant to the client (e.g., messages).

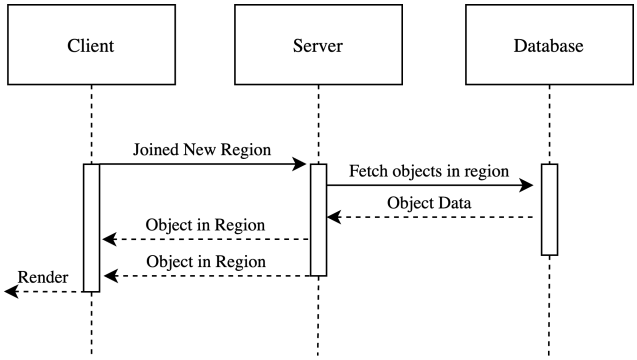

Figure 2: Joining a region with 2 objects and no other players.

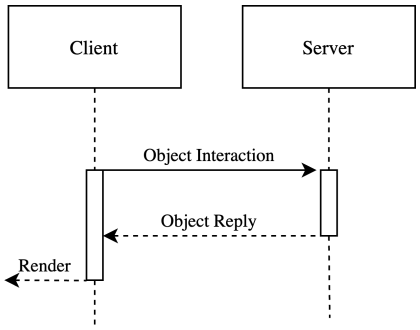

Figure 3: Interacting with an object.

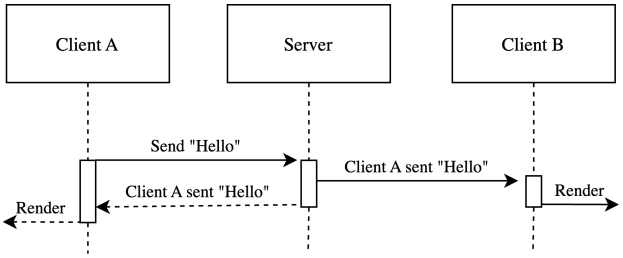

Figure 4: Sending a message to a region which is received by another player.

Habitat featured the ability to update client-stored object data "over-the-air" if objects were changed or created on the server. For

instance, an object sprite or list of interactions may be modified. The *Habitat* client software was stored on two floppy disks – one for program logic which would be stored in memory, then exchanged for the data disk with object data. Upon connecting to *Habitat*, the client announces its presence in a message which contains the version ID of the object disk. If the server has any objects that have newer version than the clients', it would transmit the relevant blocks (storing objects) which would be written to the floppy disk. Only the final update would update the version ID of the client's block, such that if the transmission failed, the client would not assume it had all the information early, acting as a kind of transaction.

Quantum Link achieved effective communication with error correction and out-of-order protection, and *Habitat* was able to utilize to provide a novel experience to its users. However, Quantum Link was not perfect. The CRC checksum was larger than necessary and took too long to compute "current method is unacceptably slow (cannot be over emphasized)" – Randall Farmer [Morningstar et al. 1986a]. There were also problems with flooding the Commodore 64 input buffers since the server did not check if the client was ready for more data. This naturally lead to more issues for users with a 1200bd connection, as more data could be sent before the Commodore 64 could process it. A bug in the Quantum Link networking meant that it would not respond to client heartbeats if NAKs were pending. According to Farmer, fixing this would allow common recoveries to be performed in 2 packets instead of 6.

4.5 Netrek

The last game we will discuss is *Netrek* (1989). *Netrek* began as *Xtrek*, a game inspired by *Empire* (1973), a strategy game for the PLATO system loosely based on the Star Trek universe [McFadden 1994]. *Xtrek* used the X Window System for UNIX on a client-server architecture, though the remote server was responsible for driving the display as well as handling server-side game logic. *Netrek* was built off of this, replacing the server-driven display with client-side rendering, and leaving the server as the controller of the world state. Both games used TCP over IP to communicate, and ran on any UNIX device with an internet connection and X10 support. In 1989, the source code for *Netrek* was uploaded to Usenet, and has remained open source since then.

Players of *Netrek* control spaceships in a large shared galaxy within which the objective is to capture as many planets for your chosen team as possible. Maximum players could be specified by the server owner, though the default value is 20. All interactions happen in real time, and the game considers itself more of a sport than a video game, necessitating features which maintain competitive integrity (e.g., cheat-mitigation, low latency, and minimal transmission error).

Historical source code, written in C, is available on Netrek.org's ftp archive, the oldest version we could find was last updated in 1992. Features added after our 1989 cutoff (UDP transmission and RSA encryption) are modularly used, so we can easily ignore them in our static analysis. By doing so, we are looking at code that is as similar as possible to the original 1989 Usenet submission. The rest of this review is based off this source code.

Netrek was designed to be flexible. Servers were designed to be easy to set up by individuals and were highly configurable through config files. Administrators could modify rules of the game on the fly, and the server would periodically read the config files and update the game without needing a restart or any interruption to the game. Since hosting was not centralized, players needed a directory to find active games. Metaservers were created, which simply hosted a static list of *Netrek* servers that could be updated by the metaserver owner. We were unable to find any evidence of metaservers in video games predating *Netrek*, so it is likely the first to use this.

Servers would also maintain records of users and their statistics in the game. Upon connecting to a server, the main thread of the server would create a thread to handle all communication to that player. Then, the user would enter a username and password to register or log in. Passwords were hashed with the standard POSIX `crypt` library, using the DES algorithm with the username as salt. If the server was full (maximum number of players already connected), connecting players could be placed into a queue, from which new players would be allowed to join when another player disconnected.

Similar to *Habitat*, the server places no trust in the information sent by the client. In complete opposition to *Maze* where clients inform the server of state changes, *Netrek* clients send requests to the server, which will update the world state if the request is valid. For instance, a player cannot set their ship's speed above its maximum, as this request would simply be ignored. The player thread on the server handles this logic, and if a request is accepted, the relevant state change is written to memory which is shared by all player threads and the main thread. This state is limited to simple information like ship speed and direction, and only the player thread writes to this area of memory. No mutexes, locks, or any other measures are used to mitigate race conditions, only the design of the software prevents them.

To update the world state with more complex data such as player position, player fuel, projectile weapon position and hit detection, or planet status, the main thread periodically reads from the shared memory and performs calculations to update this information. Therefore the main daemon handles no networking except to fork a player thread and pass it the socket used to communicate with the player.

World state is periodically sent to the player by the player thread based on a player-defined rate, the most common being 5 updates per second. In this way, the server had full control over the world state, and clients simply block until the user sends an inputs or until it is time to receive the new world state.

Connectivity would have been more robust than the earlier games we reviewed, as the game supported seamless reconnection if the client disconnected without requiring a new log in or losing the client's state. The use of TCP packets would have also guaranteed error-free and in-order transmission. Unlike in Quantum Link, the developers did not need to write any of this logic themselves as it was handled by the kernel-level implementation.

Since TCP packets were used, we will not go into the details of the overall packet structure as this is well documented. Of more interest is the structure of the packet's payload. Payloads are fairly simple, consisting of a single byte for `type`, with following bytes being additional information. Once again proving different to earlier networked games, *Netrek* has many different types of payloads instead of having a generic `DATA` payload that others have. There

are 26 server payload types, and 33 client payload types. Each is specific for one purpose, e.g., CP_SPEED is a request by the client to change their speed, and SP_PLAYER updates the player on their new location. We will not list all 59 payload types here, but instead show a few examples of them in the following diagrams.

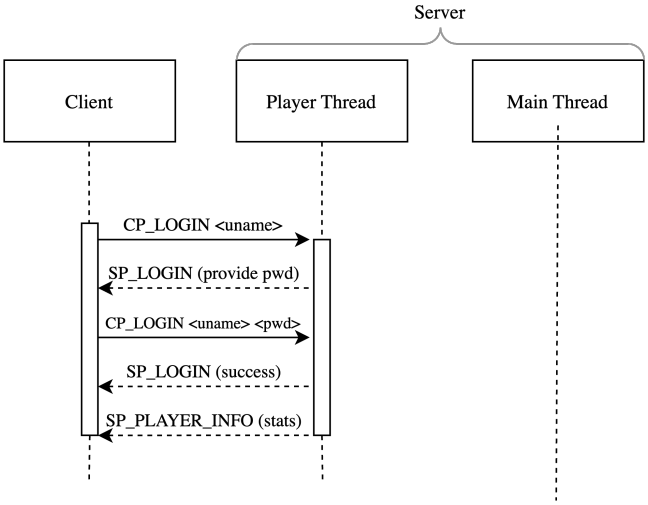

Figure 5: A user logs in with a username for the first time on that server.

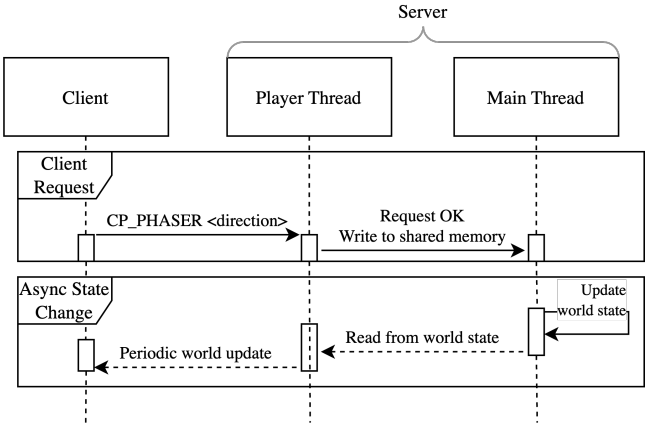

Figure 6: Typical gameplay flow. Note that the events in the lower frame can happen in any order.

5 CONCLUSION

We have described three of the earliest networked video games, detailed how they worked, and discussed the historical context they existed in as well as issues that manifested in the implementations.

Maze is notable for being the very first networked video game, targeting the IMLAC PDS-1 and using a serial connection and a server to repeat messages from a client to other connected clients.

Sopwith is a demo for BMB Compuscience's proprietary Imaginet networking software – a transparent, operating system-agnostic

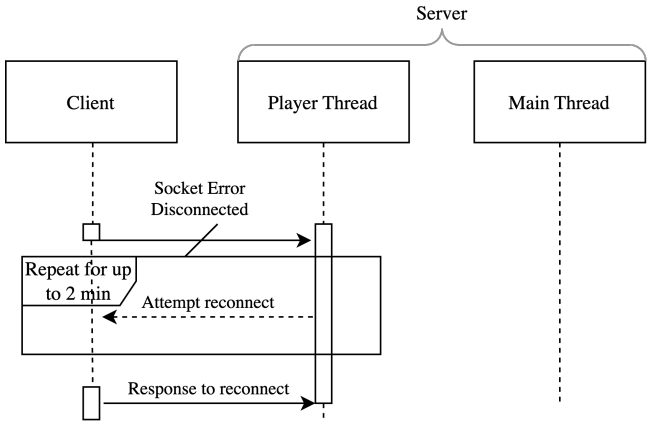

Figure 7: Reconnecting to a disconnected client.

shared drive which allowed any software to use its network by reading and writing to a virtual floppy drive.

SGI Dogfight is significant for being the first video game to use the then-nascent TCP/IP stack which is now the underlying architecture for the internet and networks which interface with it.

Networked video games gave rise to a new means for players to interact with each other, no longer confined to a single display, but free to compete or collaborate from anywhere in the world.

REFERENCES

Edward Gordon Kenneth Agnew, William M. Maclean, and Richard C. Madter. 1983. Controller and Method for Transparent Resource Sharing in Microcomputer Systems. https://www.ic.gc.ca/opic-cipo/cpd/eng/patent/1172380/summary.html Canadian Patent No. 1172380, Filed December 7th., 1983, Issued August. 7th., 1984.

John Aycock and Katie Biittner. 2023. *Archaeogaming is X.* Cotsen Institute of Archaeology Press, CA, USA, Chapter 8, 83–78.

John Aycock and Tara Copplestone. 2018. Entombed: An archaeological examination of an Atari 2600 game. *The Art, Science, and Engineering of Programming* 3 (11 2018). Issue 2. https://doi.org/10.22152/programming-journal.org/2019/3/4

Yahn W. Bernier. 2003. *Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization.* Valve Developer Community. https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization

Brian Blau, Charles E. Hughes, Michael J. Moshell, and Curtis Lisle. 1992. Networked virtual environments. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics - SI3D '92.* ACM Press, New York, New York, USA, 157–160. https://doi.org/10.1145/147156.147187

Ian Bogost and Nick Montfort. 2009. Random and Raster: Display Technologies and the Development of Videogames . *IEEE Annals of the History of Computing* 31, 03 (July 2009), 34–43. https://doi.org/10.1109/MAHC.2009.50

Brookhaven National Laboratory. 2013. *The First Video Game?* Brookhaven National Laboratory. Retrieved 2024-10-29 from https://web.archive.org/web/20160101204514/http://www.bnl.gov/about/history/firstvideo.php

W J Buchanan. 2004. *RS-232.* Springer US, Boston, MA, 239–274. https://doi.org/10.1007/978-1-4020-7870-5_11

Jack Cole. 1986. Net Working – Reflections on Network Design. https://fragglet.github.io/sdl-sopwith/imaginet.html

Tapas K. Das, Gurminder Singh, Alex Mitchell, P. Senthil Kumar, and Kevin McGee. 1997. NetEffect. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology.* ACM, New York, NY, USA, 157–163. https://doi.org/10.1145/261135.261164

Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. 1980. The Ethernet. http://ethernethistory.typepad.com/papers/EthernetSpec.pdf

Benj Edwards. 2016. *The Forgotten World of BBS Door Games.* PCMag. Retrieved 2024-12-03 from https://www.pcmag.com/news/the-forgotten-world-of-bbs-door-games

Guinness World Records. 2024. *Best-selling desktop computer.* Guinness World Records. Retrieved 2024-12-03 from https://www.guinnessworldrecords.com/world-records/72695-most-computer-sales

Simon Howard. 2024. *Imaginot.* GitHub. Retrieved 2024-10-28 from https://github.com/fragglet/imaginot

IEEE. 1985. IEEE Standards for Local Area Networks: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. https://doi.org/10.1109/IEEESTD.1985.82837

Steve Jones and Guillaume Latzko-Toth. 2017. Out from the PLATO cave: uncovering the pre-Internet history of social computing. *Internet Histories* 1, 1-2 (2017), 60–69. https://doi.org/10.1080/24701475.2017.1307544

Anker Helms Jorgensen. 2009. Context and Driving Forces in the Development of the Early Computer Game Nimbi . *IEEE Annals of the History of Computing* 31, 03 (July 2009), 44–53. https://doi.org/10.1109/MAHC.2009.41

Henry Lowood. 2009. Videogames in Computer Space: The Complex History of Pong. *IEEE Annals of the History of Computing* 31, 03 (July 2009), 5–19. https://doi.org/10.1109/MAHC.2009.53

Rob Mace. 1988. *Dogfight and others.* Unix Usenet. Retrieved 2024-10-30 from https://mirrors.nycbug.org/pub/The_Unix_Archive/Unix_Usenet/comp.sys.sgi/1988-December/009861.html

Andy McFadden. 1994. *The History of Netrek, through Jan 1 1994.* Netrek Nexus. Retrieved 2024-12-04 from https://www.netrek.org/about/history_overall.php

MediaWiki. 2024. Pywikibot. Retrieved 2024-06-26 from https://www.mediawiki.org/w/index.php?title=Manual:Pywikibot Version 9.2.0.

Robert Metcalfe and David Boggs. 2006. *The History of Ethernet.* YouTube. Retrieved 2024-10-26 from https://www.youtube.com/watch?v=g5MezxMcRmk

Chip Morningstar and F. Randall Farmer. 2008. The Lessons of Lucasfilm's Habitat. *Journal For Virtual Worlds Research* 1, 1 (7 2008). https://doi.org/10.4101/jvwr.v1i1.287

Chip Morningstar, F. Randall Farmer, and Janet Hunter. 1986a. *habitat.* Museum of Art and Digital Entertainment. Retrieved 2024-11-05 from https://github.com/Museum-of-Art-and-Digital-Entertainment/habitat

Chip Morningstar, F. Randall Farmer, and Janet Hunter. 1986b. *macross.* Museum of Art and Digital Entertainment. Retrieved 2024-12-03 from https://github.com/Museum-of-Art-and-Digital-Entertainment/macross

Michael Necasek. 2011. *DOS 3.0, 3.1, and 3.2.* OS/2 Museum. Retrieved 2024-10-28 from https://www.os2museum.com/wp/dos/dos-3-0-3-2/

Mark Nollinger. 1995. *America, Online!* Wired. Retrieved 2024-12-03 from https://web.archive.org/web/20181022022202/https://www.wired.com/1995/09/aol-2/

Howard Palmer, Steve Colley, Greg Thompson, Dave Lebling, Ken Harrenstein, and Charles Frankston. 1974. Maze. Retrieved 2024-09-08 from https://s3-us-west-2.amazonaws.com/tmmeng/Maze/Mazewar-Imlac-asm.txt

Jonathan Postel. 1981. *TCP-IP Digest, Vol 1 #2.* Usenet. Retrieved 2024-10-29 from https://groups.google.com/g/fa.tcp-ip/c/HDoq18adT2Y/m/HwoFGxcizzEJ

Jeremy Reimer. 2005. *Total share: 30 years of personal computer market share figures.* Ars Technica. Retrieved 2024-12-03 from https://arstechnica.com/features/2005/12/total-share/

Sandeep K. Singhal and David R. Cheriton. 1995. Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality. *Presence: Teleoperators and Virtual Environments* 4, 2 (1 1995), 169–193. https://doi.org/10.1162/pres.1995.4.2.169

Gary Tarolli, Rob Mace, Barry Brouillette, Marshal Levine, Dave Ciemiewicz, Andrew Cherenson, Chris Perry, Chris Schoeneman, Thad Beier, and Marc Ondrechen. 1988. SGI Dogfight. Retrieved 2024-09-08 from https://notwood.net/sgiflight/old-flight-src.zip

Greg Thompson. 2004. *The aMazing History of Maze.* Computer History Museum. Retrieved 2024-10-28 from https://www.digibarn.com/collections/presentations/maze-war/index_files/frame.html

Michael Zyda. 2000. *Networked Virtual Environments in 75 Minutes!* Calhoun: The NPS Institutional Archive. Retrieved 2024-10-29 from https://web.archive.org/web/20190417084504/https://core.ac.uk/download/pdf/36733821.pdf