

Realtime Networking Technologies for Unity

Graduation Report

RODRIGO SANCHEZ, 16.06.2020

CREATIVE MEDIA AND GAME TECHNOLOGIES
SAXION UNIVERSITY OF APPLIED SCIENCES

Information Page

Student: Rodrigo Sanchez
Student ID: 436703
Email: rodrigo_jsd@hotmail.com

Graduation company: Paladin Studios
Company coach: Olaf Abbenhuis
Guiding teacher: Paul Bonsma

Preface

This study is part of the graduation module for the Creative Media and Game Technologies course at Saxion University of Applied Sciences, the Netherlands. The assignment was conducted at Paladin Studios, a game development company situated in The Hague, between the 10th of February and the 19th of July 2020.

I would like to thank the members of Paladin Studios for allowing me to complete my graduation among them and providing their support during the quarantine period. Particularly, I would like to express my gratitude towards Olaf Abbenhuis for his extensive patience and guidance throughout the project. Likewise, I would like to thank Paul Bonsma, my graduation coach, for his support and understanding during this period. Additional acknowledgments go towards the staff of the Mirror discord, for their readiness and extensive support through the course of this project.

A quirk present in this document is the almost interchangeable use of the words *framework* and *service*. This is due to the varying nature of the ones analysed during the project. Both terms should generally be interpreted as referring to the same type of product: a middleware high-level API that provides networked functionality that can be integrated to a video game.

Abstract

The development of networked games is a complex science. Paladin Studios is interested in expanding their knowledge in this area, particularly regarding real-time multiplayer technologies. The company identifies its current networking proficiency as a limitation for exploring future opportunities.

This paper covers research into different networking topologies and message protocols appropriate for the development of a fast-paced, real-time multiplayer game with high player capacity. Sustaining a smooth game experience with these qualities incurs in various technical complexities with economic consequences attached to them which should be considered at the start of a project. In addition, there are inherent issues with multiplayer online games that affect the experience of the end-user, such as lag and cheating users; some of these can be minimized through effective netcode and adding a layer of validation for a player's actions in a server or middle communication point.

Research indicates that to achieve highly performant networked code, custom solutions are required, tailored to the type of game in question. Hence, despite the difference in the level of performance between frameworks, this is not a deciding factor – considering the developer is able to add their personal optimizations to the game. Considerably more significant is the ease-of-use of the framework, more so with inexperienced users. After their list of features, the difficulty of setup, available support, accessibility, and maintainability both in short-term and long-term projects are crucial in the selection of networking frameworks.

Table of contents

Information Page	2
Preface	3
Abstract.....	4
Table of contents	5
Glossary.....	7
Introduction	8
Reason for the assignment	9
Company Outline	9
Objectives of the Client.....	9
The question of the client	9
Products and services needed by the client	10
Expectation of the result for the client.....	10
Limiting condition and project boundaries.....	10
Preliminary problem statement.....	11
Problem Analysis.....	11
Formulation of the preliminary problem	11
Theoretical background	12
Problem Definition.....	12
Main and Sub questions.....	13
Main question:	13
Sub questions:.....	13
Deliverables.....	13
Scope.....	14
Description of the product.....	14
Results.....	15
Transport Layer	15
Network Topologies	17
Peer-to-Peer (P2P)	17
Relay Server (P2P).....	17

Dedicated Server.....	17
Cheating	18
Service analysis and comparison	19
Photon Bolt	20
Mirror Networking	20
Forge Networking Remastered (FNR)	21
Performance	22
Choosing a service.....	22
The product.....	23
Network Lag	23
Conclusion.....	26
Recommendations	27
References:	28
Appendix	30
1.1 Comparison chart by relevant framework features.....	30
1.2 Possible networking stacks to choose for Unity	30
2.1.1 Bolt Unity Editor Integration.....	31
2.1.2 Bolt Coding style	31
2.2.1 Mirror Unity Editor Integration.....	32
2.2.2 Mirror Coding style	32
2.3.1 Forge Unity Editor Integration	33
2.3.2 Forge Coding style.....	33
3.1 Discord servers compared by member count.....	33
4. Screenshots of the prototype	35
4.1 Screenshot representing a Lobby system and a headless Windows sever.....	35
4.2 AI controlled clients spawned over the network during a load test.....	35
4.3 Server metrics after a real-world scenario test with 13 clients connected concurrently.	36

Glossary

Term	Abbreviation	Definition
Application Programming Interface	API	Functions and procedures a framework or service exposes to a programmer.
High-level API	HLAPI	Directly integrated to gameplay code, API developers commonly use to work on application features.
Low-level API	LLAPI	Less abstracted than a HLAPI, it provides more granular control over underlying systems.
Server		A computer that hosts an instance of the application for users to connect to.
Client		A user that connects to a server through the network via an application.
User Datagram Protocol	UDP	
Transmission Protocol	TCP	
Internet Protocol	IP	The online address of the device. Needed to connect devices together.
Transport		Base messaging systems that read and write packages from and to the network.
Headless Server		Application stripped of graphics and other unnecessary data.
Matchmaking		Service operation that groups together clients to play a game online.
Packet		A bundle of data sent over the network.
Lag		Processing time between the start of an action and the result of it. Seen as delay.
Peer-To-Peer	P2P	Network topologies where one client acts as the host client-server of the game.
Networking Code	Netcode	A subset of code that manages communication between machines.
Authoritative		Refers to the owner of the object being allowed full control of it.
Amazon Web Services	AWS	Collection of cloud services that can be used for hosting servers.

Introduction

Multiplayer games often stand out from their competition by the robustness of the networking architecture that sustains them. Gambetta (n.d.) wrote about various issues that plague fast multiplayer games, such as cheating and network delay, as well as techniques that developers have fabricated to tackle them, such as entity interpolation, client-side prediction, and lag compensation.

The idea, however, that complex, feature rich netcode is the end solution to have good networking in any game is misguided, as stated by Olsen and Weimann (2019) who indicate, in their interview on high-performance game networking, to consider purely the usage of the necessary tools for the job. Catered to the type and requirements of a game, developers must choose between different networking architectures for client and server, transport protocol for sending messages as well as the frequency of the synchronization, a complex series of decisions.

This document opens with the definition of the assignment and reasoning behind it, the objectives of the client, and their expected outcomes, as well as the formulation of research questions and scope of the project. Subsequent sections contain the description of the products created during the research period, followed by theoretical research and the analysis of different services and the filtering method applied during the selection process, as well as the practical experience accumulated during the creation of the product. Finally, the conclusion condenses the major findings of the research in relevance to the client, while the recommendations section highlights general networking advice based on the author's learnings.

Reason for the assignment

Paladin Studios seeks to expand their knowledge of networking services and define an efficient and easy to use networking stack suitable for the creation of real-time multiplayer games. While they already possess some experience in the creation of networked games, the technology needed for a fast-paced, real-time multiplayer game is vastly different. Furthermore, numerous services are available that could prove viable for the making of one such game.

Given the reasons, the assignment consists of research and exploration of some of these services, both theoretical and practical with the goal to generate knowledge to be spread in the company.

Company Outline

Based in Caballero Fabriek in The Hague, Paladin Studios is a game development company currently composed of around thirty employees. Founded on July 1st, 2005, initially it centered on using 3D game technology for non-game purposes like 3D visualizations, big train simulators, and interior design software. In 2010, the focus of the company shifted to the creation of video games, releasing *Momonga Pinball Adventures* in the same year, which gained it international renown. In recent years, Paladin Studios has co-developed mobile free-to-play games alongside various publishers. The studio develops high-quality games for mainstream services and digital stores, from initial concept and beyond.

The previous experience in the networking field within the company is represented in *Stormbound: Kingdom Wars*, a card collection game in which players battle one versus one over the network in strategic turn-based combat. Otherwise, Paladin Studios has conducted live-ops for multiple of their released games for several months after their initial release.

Objectives of the Client

The question of the client

Netcode for games in real-time poses a complex series of issues that prevent a smooth, responsive experience for the player. The technology behind major networking structures is fairly well explained and available online. However, the implementation of said technology is an endeavor that would consume an unreasonable amount of time and effort for Paladin Studios.

The client would like to know what the most appropriate networking services are for the creation of a real-time multiplayer battle royale type of game. The criteria for what makes a service good is further defined in this document, first-and-foremost of which are ease-of-use and efficiency; reinventing the wheel requires much effort, and thus, a rich number of ready-made features is of great importance when choosing a service.

Products and services needed by the client

The client has requested that the solution be compatible with the Unity game engine and, ideally, to allow hosting the game server on a Linux machine and using Amazon Web Services (AWS), to reduce the overall costs of deploying the game. There are no more specific third-party products required, the exploration of them is part of the assignment.

Of the student, the client wishes to receive a comprehensible guide based on the conducted research, which gives an overview of important concepts of real-time networking, useful techniques, and a catalogue of third-party services, their strengths and weaknesses summarized. Moreover, prepared workshops on the topic for the developers in Paladin would be a preferred outcome of the assignment. An ideal outcome would include the creation of a framework that developers could easily import in their projects to start prototyping networked games.

Expectation of the result for the client

The client intends to utilize the knowledge generated during this assignment to provide developers in the studio an easy way to prototype new game ideas that may need the use of a real-time multiplayer framework, as well as to fill in a knowledge gap in the skill set of the studio.

Limiting condition and project boundaries

The project must abide by a non-disclosure agreement on internal matters, mention of other services in current use by Paladin, and the sharing of any code or unreleased content from any previous or ongoing game projects. Due to this, the assignment is designed with the intention of it being a standalone project. Other limitations include the development time of five months, working two to three days per week on the assignment, and the budget needed to work with some of the networking services in question.

Preliminary problem statement

Problem Analysis

Paladin has lacked a general knowledge or specialists with proficiency in networking for real-time multiplayer games. The knowledge gap makes it impossible to create games or quickly start prototypes that need these technologies, severely limiting the possibilities of new ideas the studio may come up with. Additionally, real-time networked games make up a considerable share of the market in the game industry; until this gap is reduced, the company is essentially shut off from a large volume of opportunities in business development. Most publishers will demand to see evidence of previous work or to otherwise validate that Paladin can carry out the development of a project that needs such skills. As a matter of fact, while there are individuals versed in the subject, there is scant proficiency in networking within the company. While the success of this project is not critical for the studio, it would provide substantial benefits that could lead to the acquisition and development of a new game project in accord with a publisher.

Within the project itself, the analysis and assessment of the ease-of-use of different platforms and services requires qualitative measurement. Likewise, ensuring the efficiency of messages sent over the network is fundamental for a fast-paced game. More importantly, the chosen framework must support a responsive and smooth experience for the player. These aspects must be tested and catalogued in an easy-to-follow format.

Formulation of the preliminary problem

The lack of experience in networking games at Paladin severely limits the creative, technical, and financial potential of the studio. Resolving this issue by spreading knowledge to the developers through curated research and workshops, or by providing them with an “easy-to-jump-into” custom networking framework would achieve the most success for the client.

Theoretical background

Game netcode resources on the internet are scarce in comparison to other areas of development. Most existing articles only cover a general idea, the basics of the topic, introductory notes or focus only on the low-level messaging protocol. Attempts to find more in-depth explanations or implementation details are hard to find. Olsen and Weimann (2019) mention that after the basics, there are not a lot of resources for high-quality netcode, and that most good resources on the techniques used these days are dated.

Nonetheless, several valuable resources can still be found on the internet. In *Multiplayer Game Programming*, Glazer and Madhav (2015) define many fundamental concepts of game networking in a clear way. Gambetta (n.d) publicized a series of articles on fast-paced multiplayer games in which he details important aspects of their architecture; likewise, an article on the networking behind the Source Engine is available to the public, covering similar topics to Gambetta's (Valve software, 2005). A couple of sources on more complex concepts are also available online, such as floating-point determinism (Fiedler, 2010) and rollback (Infil, 2019).

Suitable for the comparison and filtering of networking frameworks, comparison charts, blogs, and forums can be found online, compiled by unknown individuals. Additionally, netcode analysis of popular games ("Battle(non)sense", n.d.) and benchmarks of low-level networking transports ("Nxrightthere/BenchmarkNet," n.d.) have been made available by skilled community members.

Apart from these specific resources, a wealth of knowledge can be gathered while exploring forums in various developer portals and communities, presented in a much less organized manner. Of note, the wikis, documentation pages and Discord communities of the networking frameworks to be researched are invaluable for the development of this project.

Problem Definition

The problem that the client would benefit most from solving is to fill the knowledge gap that the company's developers have with networking. Unfortunately, achieving this raises multiple subproblems, like defining a good medium to share this knowledge. It is more important, however, to choose an appropriate networking framework based on the range of features it offers and the ease to pick up by developers at Paladin Studios. The combination of the right tools for the task and a jump-start on networking competence will ensure that Paladin is able to easily start new multiplayer prototypes and bring them to fruition.

Main and Sub questions

Main question:

What is the most accessible and efficient, currently available networking framework that the developers in Paladin Studios can use for the creation of a battle royale type of game for mobile platforms using the Unity engine?

Sub questions:

1. What model of network architecture is ideal for hosting fast-paced multiplayer games?
2. What development techniques are necessary to effectively host a game with many concurrent players?
 - 2.1. What are the technical and economic consequences of hosting a game with many concurrent players?
3. What are the main issues present in real-time multiplayer games and how do they affect players?
 - 3.1. What are the negative effects of network lag and how can they be reduced?
 - 3.2. How can cheating be prevented in an online game?
 - 3.3. What are some techniques developers can implement into netcode to improve the perceived responsiveness of a game for players?
4. How can different networking services be compared with each other?
 - 4.1. How can the ease-of-use of a networking framework be measured?
 - 4.2. How can the efficiency of a networking framework be measured?

Deliverables

- Proof of concept prototypes that replicate a small game including the necessary technical functionality and an overview of their difficulty of implementation.
- A workshop-type presentation and attached relevant resources. This is to be conducted at the end of the graduation period for Paladin Studios in order to condense and transmit the learnings of the project.

Scope

The objective of the graduation assignment is not to create a fully-featured game nor a final solution containing all features a network framework can have, but to create a proof of concept focused on proving and testing the given technologies. Furthermore, due to the limited time period of this project, the creation of features is discouraged over research and testing of existing tools and services. For this reason, it is not necessary to implement advanced techniques of networking such as rollback or compression algorithms, unless a framework offers them out of the box.

While the assignment includes the proposal of a complete technological stack, the focus of the research is in mid- and high-level frameworks. Low-level messaging systems, backend services such as matchmaking, and deployment of game instances to the cloud, including server orchestration, are out of scope for this project. Finally, the game needs to run on Windows, Android, and iOS on cross-platform networking; Consoles and other platforms will not be necessary in these tests.

Description of the product

At the start of the graduation period, the client discussed the desirable characteristics and functionality to be proven possible to achieve with a service as a result of the project. As such, the base requirements for the prototype delivered are as follows:

- A real-time multiplayer game that can handle as many players as possible.
- Supports cross-play between PC, iOS, Android.
- The state of the game and objects are identical between players.
- Handles player reconnection, returning them to their state before disconnection.
- Movement is synchronized and is smooth even in high latency situations.

The prototype was to be created using the Unity game engine. The design of the game had to allow for easy analysis and confirmation of the intended behaviour. With this motive, the prototype features a battle royale game mode, simulating a boxing match – last player alive is the winner. The game mechanics are simple, but they present the core cases of scrutiny:

- Constant actions: Character movement and rotation.
- Instant actions: Punching.
- Persistent state: Player name and color.

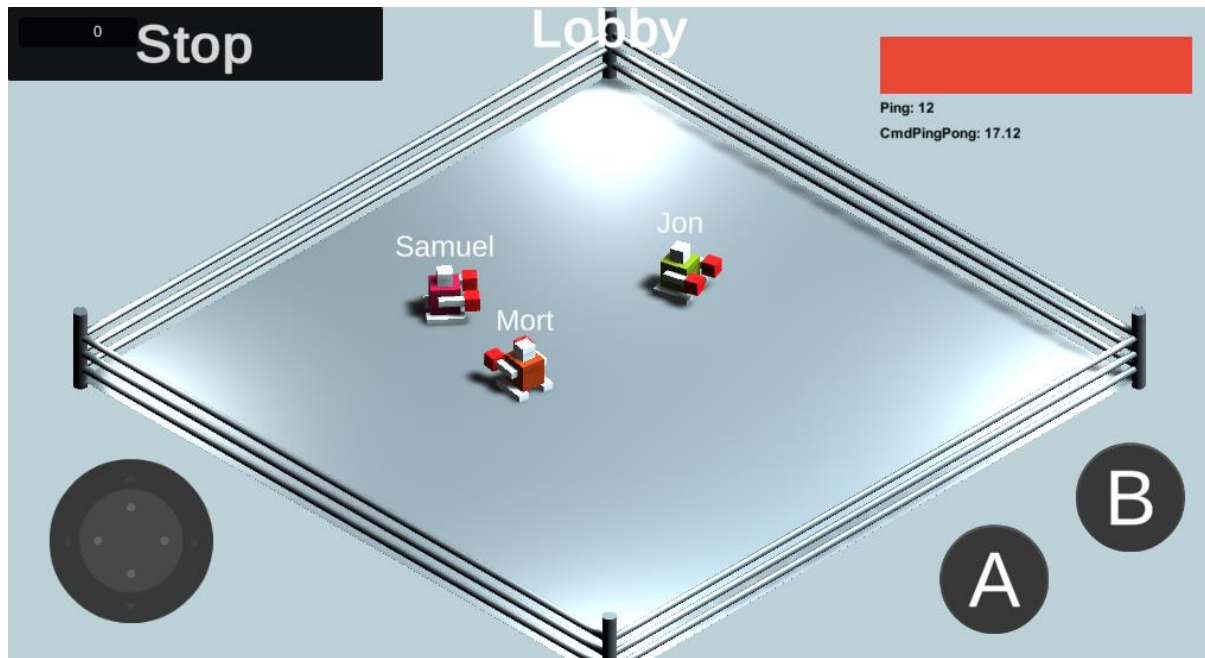


Figure 1. An early screenshot of the game prototype.

These cases allow for the observation of correct state synchronization and latency, as well as their scalability as the number of concurrently connected clients is increased. Tests during development has been carried out primarily on a Windows PC, utilizing separate game builds for server and client, connected over the external network.

Results

In order to conduct a proper analysis of frameworks, understanding fundamental concepts of networking and their use cases is compulsory. Literature analysis from various sources provided insight from experts as well as the community, findings which aid to answer many of the sub-questions of this research.

Transport Layer

A library of methods that serves as a bridge between the application code and messages sent across the network is commonly known as the transport layer (Ubiquity Networks, n.d.). They determine how, when, and what sockets are used to send and receive data packets over the network. These parameters are controlled by the transport protocol used to send a message; the two most commonly used protocols are TCP and UDP. First and foremost, the low-level network protocol to prioritize for the research must be determined, as not all services are built on, or support using more than one transport protocol.

Each network packet is broken into a section that contains the protocol's parameters for sending the message, called a *header*, and the meaningful information of the packet referred to as the *payload*. Since the payload is the same regardless of the protocol used, the difference falls mainly in the header.

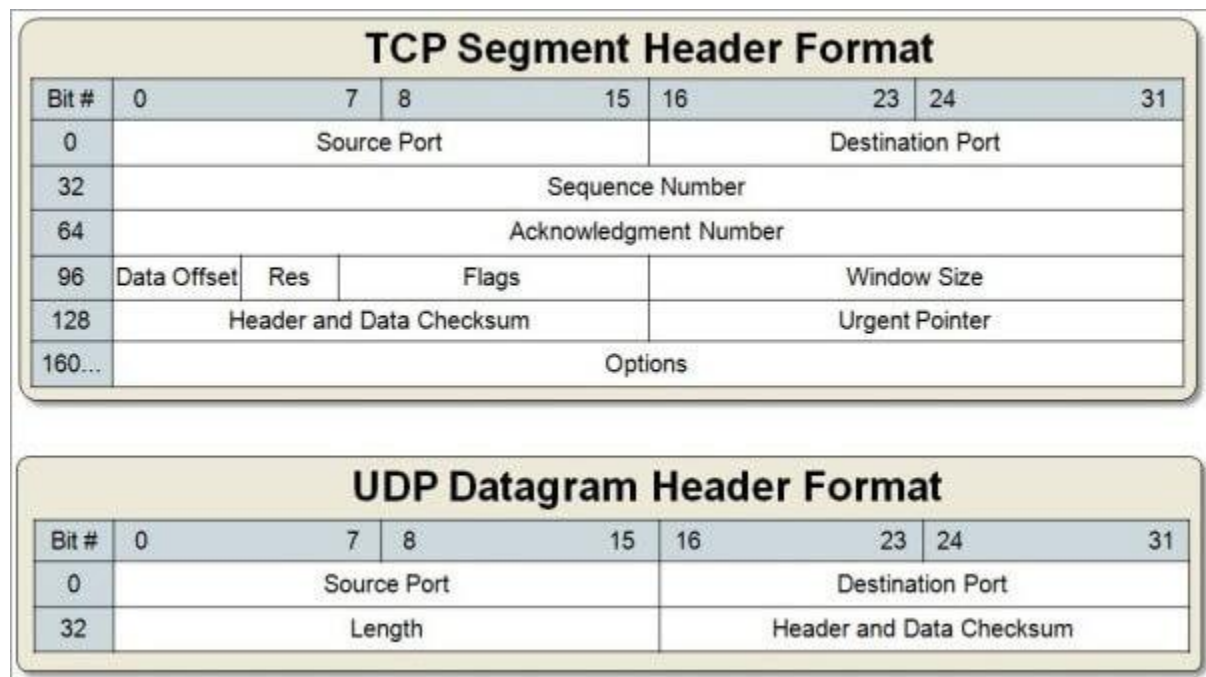


Figure 2. The structure of a TCP and UDP headers. (Software Testing Help, 2020)

In figure 2, the TCP protocol guarantees reliability and correct ordering of messages, alongside other features. Its drawbacks include slower processing speed from the overhead caused by the extra functionality and the increased package size due to a larger header. In contrast, data sent through UDP is unreliable — there is no flow control, error checking, and packages may be reordered or even dropped.

Real-time games will usually prefer sending data over UDP due to the packages being lightweight and not incurring the increased latency of TCP (Purdum, 2013). Some transport libraries minimize the risks of UDP's unreliability by providing optional flags that simulate the reliability of TCP (SoftwareGuy, 2020). This mode of sending data is known as reliable UDP or rUDP.

Network Topologies

Connecting users over the internet can be achieved in different ways. In 2018, House discussed the benefits and drawbacks of different network topologies:

Peer-to-Peer (P2P)

This model has clients connect directly to one another. Typically, one of the clients serves as the host for the match. The use of this model has decreased through the years as it implies making too many sacrifices — the connection quality is reliant on customer hardware, leading to widely varying latency. Additionally, it faces major security issues due to players being able to see other players' information and manipulate the game to cheat. Finally, if the host leaves the game, the game ends for everyone, unless developers support host-migration, a complex feature to implement (Battle(non)sense, 2017). One of the few real benefits of this architecture is that it tends to be cheaper than others, as it incurs in no financial fees for server hosting.

Relay Server (P2P)

A technique similar to peer-to-peer, except that network messages must first go through a separate server that relays the packages to players. This can help add an extra layer of security and expand the network reach but since messages need to make more hops around the internet before reaching their destination, it increases the total latency of a game.

Dedicated Server

By far the most expensive option, it is also the one that offers the most benefits provided it is implemented correctly since the level of complexity is higher over other solutions. With sufficient coverage, dedicated servers can achieve very low levels of latency and the best possible security, preventing cheating and keeping user information private, all while being available to more people around the globe. The economic cost of dedicated servers is directly related to the size and locations of the audience playing the. In 2018, House discussed the difficulty of implementing dedicated servers for a game considers the addition and integration of various other pieces to the machine: separation of client and server, multiple backend services, and development of features that ensure server authority.

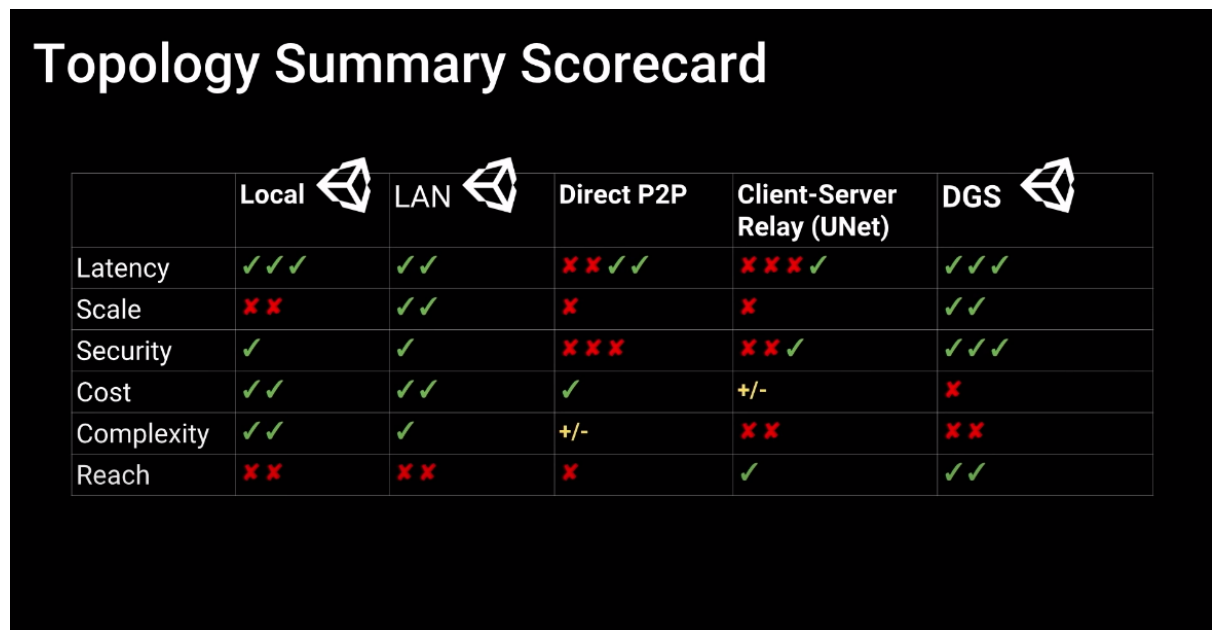


Figure 3. A chart comparing factors of different topologies in relevance to real-time games. (Unity, 2018, 03:00–13:21)

Cheating

A commonplace saying in the networking world is to “never, ever trust the user”. This is attributed to the certainty that players can and will exploit any vulnerabilities of an application to gain a personal advantage. This is particularly problematic for competitive games like first-person shooters and MOBAs. Instances of cheating can be seen as hacking an application and altering it or injecting custom code on top of it, with which a user may change their avatars to be faster, invincible, or have access to items that other players commonly don’t.

By far the easiest way of preventing users cheating is to enforce an authoritative server architecture (Gambetta, n.d.). Only possible with dedicated servers, every action a client wants to perform must be first validated by the server. The simulation and state are maintained in the server and sent to every client. Unfortunately, this architecture generally results in delays between user input and the desired action being performed, which may result in a terrible experience for the user. While there are ways to resolve this issue, such as client-side prediction and rollbacks, they add an increasing level of complexity to the netcode.

Peer-to-peer networks require complex validation methods to minimize cheating, but since the code is run on each user’s machine, it is still possible to tamper with it. Client-server architectures utilizing a relay server provide an extra layer of security, and some services allow for cheating prevention code to be added in the relay.

For a fast-paced action game in networked real-time, dedicated servers would prove to be an ideal choice due to the possibility for minimal latency as well as high security. Most of the cost and complexity drawbacks of them are not in scope for this research, and Paladin Studios considers them manageable risks, making this a viable topology for this research. Since dedicated servers would provide a better experience in a production environment, a networking service that supports this topology is to be preferred.

Service analysis and comparison

There is an abundance of networking services available at the time of writing, and many promising ones scheduled to become available by 2021. Each of these products offers different features and are placed at different price points. After research and collection of various candidates, an initial pruning phase supported by previous comparison user-made documents, such as the ones created in 2019 by Serinx and Mikson. This filtered out misplaced services and those with low parity on the desired features or that proved unrealistically expensive. A comparison table considering relevant aspects of the remaining services was created in order to grade them through a high-level overview. This table can be found on appendix 1.1.

In consultancy with the company supervisor, three of the most promising services were selected for deeper exploration. The decision was made to consider frameworks of different categories and that fit the project's schedule. These are *Photon Bolt*, *Mirror Networking*, and *Forge Networking Remastered*, all of which support a dedicated server topology and a UDP messaging system.



To further assess these frameworks, two categories were established as points of analysis:

- **Feature set:** The out-of-the-box capabilities of a framework, advanced features, and support for expandability.
- **Ease-of-use:** The learning curve and ease of development using a framework, as well as the amount and quality of available documentation and developer support.

The following are summarized descriptions and analysis of the selected frameworks, further details about each can be found on the appendix section, as well as companion documents.

Photon Bolt

Part of the Photon networking product stack, Photon Bolt is a paid closed-source service with an impressive set of out-of-the-box available features, primarily focused on fast-paced action games.

Feature Set

A leading service in this category, Photon Bolt provides advanced tools that fit the target game-type well. These include authoritative movement, lag compensation, hitbox recording, client prediction, interpolation, and more (Exit Games, 2020). Photon Bolt requires the usage of the underlying Photon Realtime cloud services owned by Exit Games.

Ease-of-Use

Setup is more complicated than the other frameworks selected, as it involves creating an account and registering the game instance in a browser beforehand. Photon Bolt requires the use of Unity editor graphical interfaces to set up and manage all networked objects. After editing these the user needs to press the *compile* button in order to start the code generation process. After these steps, the developer can extend the generated boilerplate classes via code to implement the game logic.

The documentation and API are extensive and easy to read, and while customer support is not centralized, appropriate developer and peer support can be found on its Discord server and forums.

Mirror Networking

A community-based open-source networking library, Mirror was initially developed as a replacement and improvement over Unity's UNET deprecated networking solution (Mirror Networking, n.d.).

While it does not include as many ready-made features as Photon Bolt does, it offsets this advantage through their much larger and supportive community.

Feature Set

The Mirror package includes various components for synchronization and interest management. It is targeted for massively multiplayer online games. Mirror allows for extensive customization to refine the engine to the game being developed, including easily swapping of the transport layer for other supported libraries. The active amount of concurrent developers guarantee new official updates once a month to the Unity Asset Store (Mirror Networking, n.d.). Additionally, it is the only service that mentions test coverage as part of the suite, sitting at 50% at the time of writing.

Ease-of-Use

Easy to setup, and presenting good, extensive documentation and tutorial videos. The API documentation is easy to navigate through as well. Easily its largest benefit, the active community of users and developers in their Discord server provide extremely helpful and quick support, with response times within one or two hours. Since Mirror was designed as a replacement of Unity's UNET, the workflow is very familiar to Unity developers, requiring minimal additions and changes to network a gameplay class.

Forge Networking Remastered (FNR)

Similar to Mirror, FNR is an open-source networking library, however, its workflow resembles that of Photon Bolt's more closely. As such, it could be considered a middle point between the two and a good candidate for this research.

Feature Set

One of the main advantages FNR presents over other frameworks is that it is engine-agnostic, which means it is not bound to the Unity engine. Developed and maintained by two main developers, FNR has a two-year release window for major updates. Additional features provided with the framework include a NAT-punch through server, state rewinding and Master servers.

Ease-of-Use

Presenting good quality but sometimes scarce documentation, FNR fails in this category by providing some outdated video tutorials, and a hard-to-read API. Technical support is decent in their Discord server, though it is mostly provided by an experienced user rather than the developers. Developing with FNR involves frequent use of the Forge Wizard in the Unity editor to generate boilerplate classes for each object which the user must then extend with their custom logic. FNR makes use of threading, a great feature, however due to its implementation, user code generally includes cumbersome code that must be used to switch between threads.

Figure 4. An ex

```
public override void Move(RpcArgs args)
{
    // RPC calls are not made from the main thread for performance, since we
    // are interacting with Unity engine objects, we will need to make sure
    // to run the logic on the main thread
    MainThreadManager.Run(() =>
    {
        transform.position += args.GetNext<Vector3>();
    });
}
```

Performance

Referring to the usage, the memory footprint, and network performance of an operation, this is an important concept for the maintenance of servers. To properly optimize an application, it must be tested in its ideal state and target environment. While networking frameworks carry some overhead, the main detriments to a game performance are found in the application developer's code. Due to this, it is important to plan ahead what data, and how and when it is sent over the network (Weimann, Olsen, 2019). The following table presents information of the performance of different transportation layers, put under stress tests. Worthy of note in the list, ENet is a UDP library that can be used as a transport system by Mirror, while Bolt uses the Photon as data transport, displaying poorer results than its competitors in this benchmark.

1000 clients

GC mode: Server

Networking library	Protocol	CPU usage (Max)	RAM usage (Max)	Transfer (Total)	Status
ENet 2.0.8	UDP	14%	56 MB	177 MB	Passed
UNet 1.0.0.9	UDP	20%	188 MB	200 MB	Passed
LiteNetLib 0.8	UDP	24%	240 MB	200 MB	Passed
Lidgren 1.7.0	UDP	15%	225 MB	232 MB	Passed
Photon 4.0.29	UDP	40%	140 MB	240 MB	Passed
Neutrino 1.0	UDP	20%	255 MB	160 MB	Passed
DarkRift 2.2.0	TCP/UDP	28%	74 MB	150 MB	Passed

Figure 5. Server benchmarks of various UDP transport layers (nxrighthere/BenchmarkNet, n.d.).

Choosing a service

Simple prototyping with each framework was conducted alongside the aforementioned comparisons. The basic prototypes were built on top of a common base of the final product and required adding networking to spawning players, synchronizing movement and collision checking. The hands-on experience was essential to determine the difficulty of set up and learning curve and development workflow with each service.

When compared to Photon Bolt, both Mirror and FNR are well documented, and though Bolt adds extra complexity during the initial setup, working with it is mostly straightforward, aside from its abundant use of additional interfaces.

Since Mirror and FNR are in the same category as free and open-source solutions, and because Mirror has an advantage with its ease-of-use in every regard, I consider Mirror to be the better choice given the project's goals.

While Mirror's feature set is strictly inferior to Photon Bolt's, it is completely free and open source. This, in addition to its much larger community, makes it a more appealing choice over Bolt for a user that is experienced in the networking area. Additionally, since the framework allows for it, it can be expanded to add features similar to Photon Bolt, given the time and effort to do so, which would make it a viable solution for a production environment. Since one of the main targets of this project is to facilitate the acquisition of knowledge regarding networking, Mirror is a great framework for developers familiar with Unity but new to multiplayer, while keeping open the possibility of usage for real-world productions.

The product

The proof of concept game developed with Mirror Networking includes the following highlights:

- **Lobby System:** Creates an area between the offline scene and game scene where connected players can see each other. The lobby system manages the transfer of data from the room objects to the gameplay objects.
- **Reconnection:** A server is able to store a disconnecting player's data and re-apply the state upon successful reconnection, as long as the player used the same login key during startup.
- **Matches:** Full health system, scoring system, and match loop;
- **Target platform testing:** Builds for the target hardware were created to ensure correct functionality in an ideal environment. Windows and Android clients are able to connect to a headless Linux server instanced on a separate machine.

On the other hand, there were desirable features that went unimplemented, the most relevant of which was server authoritative movement. This feature prevents clients from cheating by giving the server control over all objects in the game. To support the reasoning behind the exclusion of this feature, one of the most prominent issues that networked games face must be presented: latency.

Network Lag

Network latency or lag is the roundtrip time between a message going from client to server and then back. This includes the hardware processing time in the client and server machines. When dealing with Netcode, one can never assume that a signal can get from A to B instantaneously, and in fact, a large proportion of netcode exists simply to deal with this issue (Meseta, 2019). If this latency is unaccounted for, two different machines can lose synchronization of the game state, leading to a different outcome.

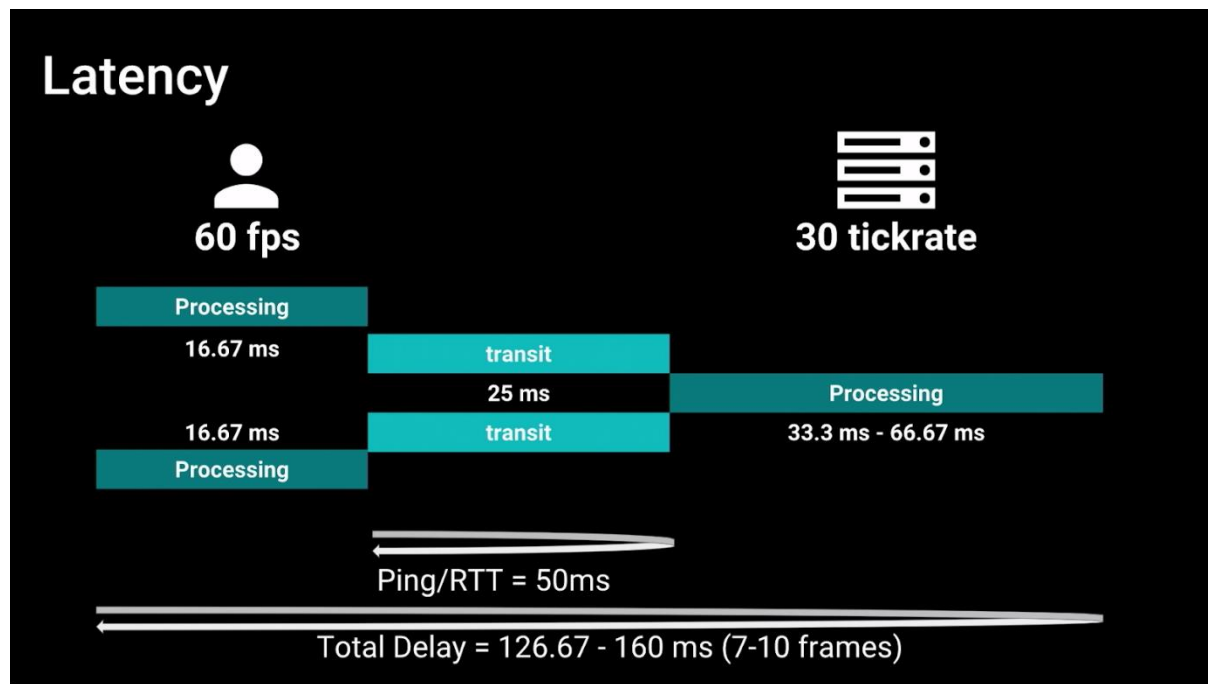


Figure 6. An explanation of the total network lag a user may experience. (Unity, 2018, 13:54–17:29)

Dealing with Perceived Lag

Network lag is a real concern; however, users often attribute other factors not related to the internet to network latency. This comes to question if there is a unified definition of latency. Players may refer to as “lag” to the delay between their inputs and receiving a response from the application. The refresh rate of a monitor may also affect how quickly a changed image is rendered for the player. While these are out of reach for developers to fix or improve, there are ways to reduce the perceived lag of a game by tweaking the design of it. Goyette describes how character animations and speed were changed to successfully reduce the perceived latency players felt in *Call of Duty: Black Ops III* (Activision, 2016).

Responsiveness is one of the main key-performance indicators for the proof-of-concept. Server authoritative movement adds latency between player input and the action being performed in the client, as the action needs to travel over the network before being executed. A second reason for excluding this feature is that to solve this latency issue requires the development of a larger subset of complementing techniques of increased complexity, that the project would not allow time for. Some of these techniques are hereby described for completeness based on Fiedler’s and Gambetta’s articles on the topics.

- **Client-side prediction:** As the client sends inputs to the server, it will also execute the logic locally, providing instant response to the player. This is corroborated, and corrected if needed, with the information later received by the server.

- **Interpolation and extrapolation:** Specifically needed to smooth out the effects of client prediction, as on its own, it can cause objects to teleport around, breaking suspension of disbelief. These techniques slowly interpolate the client state to the real server state over short periods of time.
- **Rollback:** Most prominent in fighting games or games with projectiles, rollback code can ensure the quality of hit detection. This process follows going back in time and reapplying the correct inputs sent by the server, replaying the game simulation frame by frame to return it to the correct state (Heart, 2019).

Another area of the product that could have been developed further is the optimization of network performance. Every second, various data packets are sent between server and clients, which increases exponentially based on the number of concurrent players in a game.

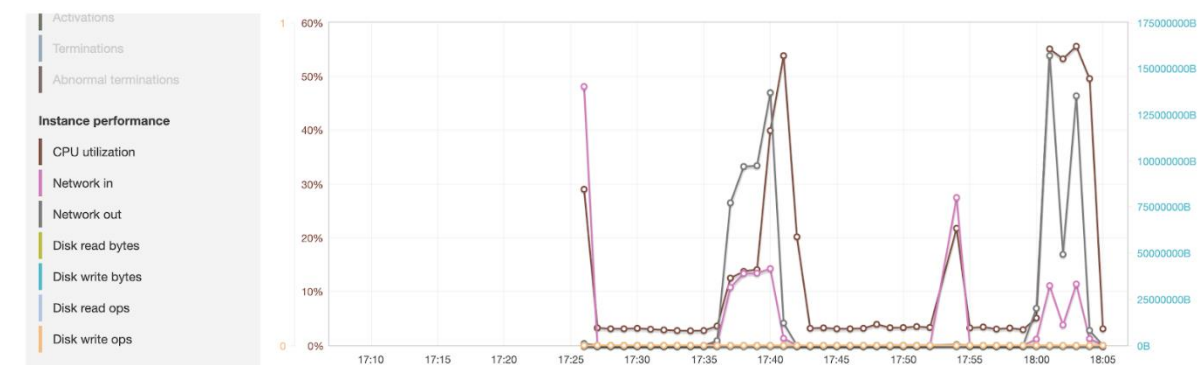


Figure 7. Network data utilization on a load test using thirty AI clients connected to an AWS server.

Load tests conducted with artificial players designed to stress the server showed to use high amounts of bandwidth, which caused network congestion between server and client machines, resulting in lag and disconnections. Based on these tests, a safe number of concurrent players in the proof-of-concept game should not exceed twenty in order to provide a decent user experience.

Reducing the bandwidth required per player would necessitate further research and analysis to utilize more efficient methods of sending data over the network, as well as a re-design of the product to account for the least amount of required data sent per second. Alongside the increased time debt of it, this process would introduce extra complexity to the codebase, rendering it harder to read – an undesirable effect for a project meant as example.

Finally, a user test conducted with thirteen of Paladin Studios employees connected concurrently to an external server showed positive reactions. Users communicated good responsiveness and low perception of lag, but they expressed their feedback on the inconsistencies of the server-authored collision checks when punching other players. Network usage of this test is found on Appendix 4.3.

Conclusion

Fast-paced real-time games benefit from sending data primarily over UDP due to it being lightweight and faster than TCP. While it is heavily dependent on the type of game, dedicated servers are generally the best choice for a network architecture that can result in better player experience. However, dedicated servers imply increments in development complexity and deployment costs over relay and P2P networks.

Latency is the major issue players commonly face with networked games. Developers can reduce latency by profiling and optimizing what data they send, the size of it, and the rate of sending to avoid consuming too much bandwidth is used. Most of the time, however, studios need to add complex algorithms to their games such as client-side prediction, rollback, and interpolation, as well as sometimes changing the design of the game to improve the perceived responsiveness of the game for users.

Selecting one networking service over another is a difficult task where personal preference plays a part. While performance comparability is certainly important, most games that reach official production will necessitate the developer to perform custom optimizations tailored to their game. So, it is a less relevant comparison point as the feature set of a service, or its ease of use.

Creating networked games is a difficult task, so of the two points, the latter is crucial for developers that are not as comfortable in the field. Mirror Networking is a great framework and a good fit for Paladin Studios at this time due to the simplicity of the API, excellent support, and documentation. In addition, heading into production with it is feasible, thanks to its expandability and customizability. Photon Bolt, on the other hand, offers advanced functionality out of the box, if one can afford it. At the start of a new project, developers must consider the requirements of their game before choosing an appropriate networking stack that will result in the least amount of compromises for the studio.

Recommendations

Before starting the development of commercial or serious productions, it is imperative for the success of the project that the users behind it have an appropriate understanding of networking to determine the best fitting networking architecture for their specific use case. Besides laying out the requirements of the game mechanics, a designer or producer should consider the complexity of the networking stack required to support the type of game in question, as this factors heavily in the costs and risks of the project.

To make a proper assessment of the networking stack required for a project to perform appropriately, I would recommend for the developers of the project to start by making small projects and testing them so they can become acquainted with the new workflow. As is discussed in this paper, the addition of networking to a game adds a new level of complexity to that forces the developer to face many new hurdles not present in single player-games, such as keeping game state synchronized and maintaining responsiveness. While an ideal situation would be for everyone in the team to have a decent experience with game networking, a more realistic recommendation is for a small group to become educated in the topic, or to hire a professional with adequate qualifications.

As an additional note, networking requires extensive testing for bugs and quality of performance. I would advise to directly double a normal time/cost estimate during project planning, to ensure that there is sufficient room for these tests and fixes to take place.

References:

Glazer, J., & Madhav, S. (2015). *Multiplayer Game Programming* (1st ed.). Harlow, United Kingdom: Pearson Education.

Gambetta, G. (n.d.). Client-Server Game Architecture - Gabriel Gambetta. Retrieved March 16, 2020, from <https://www.gabrielgambetta.com/client-server-game-architecture.html>

Valve software. (2005, June 30). Source Multiplayer Networking. Retrieved March 16, 2020, from https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

Fiedler, G. (2008). Game Networking. Retrieved March 16, 2020, from <https://gafferongames.com/categories/game-networking/>

Weimann, J., Olsen, K. (2019, December 21). *High Performance Game Networking in Unity3D + Q&A - (submit questions early)* [Video file]. Retrieved from https://www.youtube.com/watch?v=5b6k_ywdjw4

Battle(non)sense. (2017, August 29). *Netcode 101 - What You Need To Know* [Video file]. Retrieved from <https://www.youtube.com/watch?v=hiHP0N-jMx8>

Serinx. (2019, January 7). Unity Multiplayer - What are the pros and cons of available network solutions/assets. Retrieved from <https://forum.unity.com/threads/what-are-the-pros-and-cons-of-available-network-solutions-assets.609088/>

Mikson, J. S. (2019). *Unity Networking Frameworks - Feature List* [Comparison of networking middleware]. Retrieved from <https://docs.google.com/spreadsheets/d/100vNy3grUgLV5M7rIUkUtRqO4cmTewQI8P5uwf-Qb9k/edit#gid=1012362019>

nxrighthere/BenchmarkNet. (n.d.). In *GitHub*. Retrieved March 16, 2020, from <https://github.com/nxrighthere/BenchmarkNet/wiki/Benchmark-Results>

Infil. (2019, October 16). Netcode [p1]: Fightin' Words. Retrieved March 16, 2020, from <http://ki.infil.net/w02-netcode.html>

Ubiquity Networks. (n.d.). Intro to Networking - Transport Protocols & Network Ports. Retrieved May 25, 2020, from <https://help.ui.com/hc/en-us/articles/115006614247-Intro-to-Networking-Transport-Protocols-Network-Ports>

Software Testing Help. (2020, April 20). TCP Vs UDP – What Is The Difference Between TCP And UDP. Retrieved May 27, 2020, from <https://www.softwaretestinghelp.com/tcp-vs-udp/>

SoftwareGuy. (2020, April 12). SoftwareGuy/ENet-CSharp. Retrieved May 27, 2020, from <https://github.com/SoftwareGuy/ENet-CSharp/blob/master/DOCUMENTATION.md>

Unity. (2018, November 19). Connected Games: Building real-time multiplayer games with Unity and Google - Unite LA [Video file]. *YouTube*. Retrieved from <https://www.youtube.com/watch?v=CuQF7hXIVyk>

Activision. (2016, August 30). *Fighting Latency on Call of Duty Black Ops III* [Video file]. Retrieved from <https://www.gdcvault.com/play/1023220/Fighting-Latency-on-Call-of>

Purdum, N. (2013, August 1). How Quake came to one of the world's first online game services. Retrieved May 16, 2020, from https://www.gamasutra.com/view/news/197460/How_Quake_came_to_one_of_the_worlds_first_online_game_services.php

Exit Games. (2020, June 16). Bolt Overview | Photon Engine. Retrieved May 16, 2020, from <https://doc.photonengine.com/en-us/bolt/current/getting-started/overview>

Mirror Networking. (n.d.). vis2k/Mirror. Retrieved April 23, 2020, from <https://github.com/vis2k/Mirror>

BeardedManStudios. (n.d.). BeardedManStudios/ForgeNetworkingRemastered. Retrieved April 23, 2020, from <https://github.com/BeardedManStudios/ForgeNetworkingRemastered/wiki/Basic-RPC-Example>

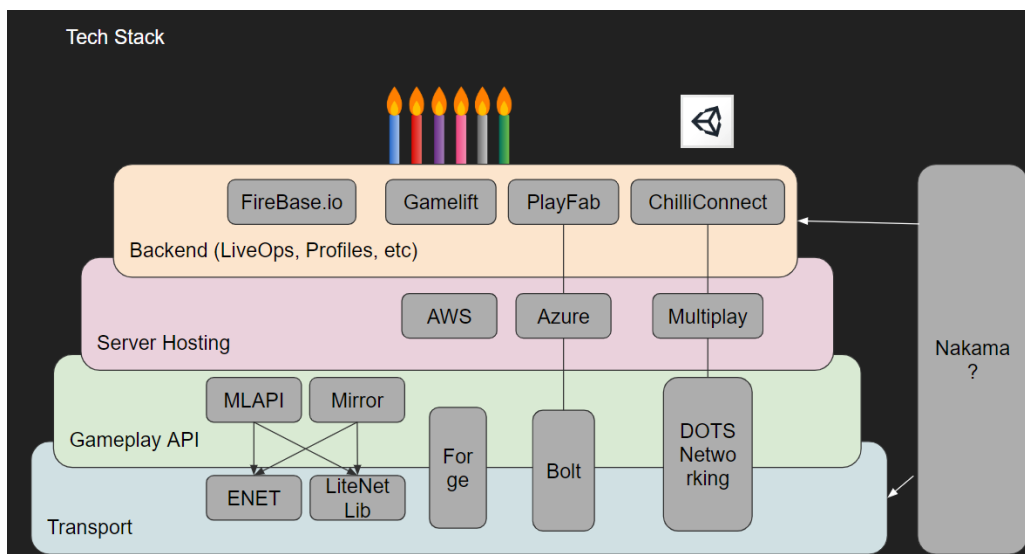
Meseta. (2019, September 22). Netcode Concepts Part 1: Introduction - Meseta. Retrieved May 12, 2020, from <https://medium.com/@meseta/netcode-concepts-part-1-introduction-ec5763fe458c>

Appendix

1.1 Comparison chart by relevant framework features

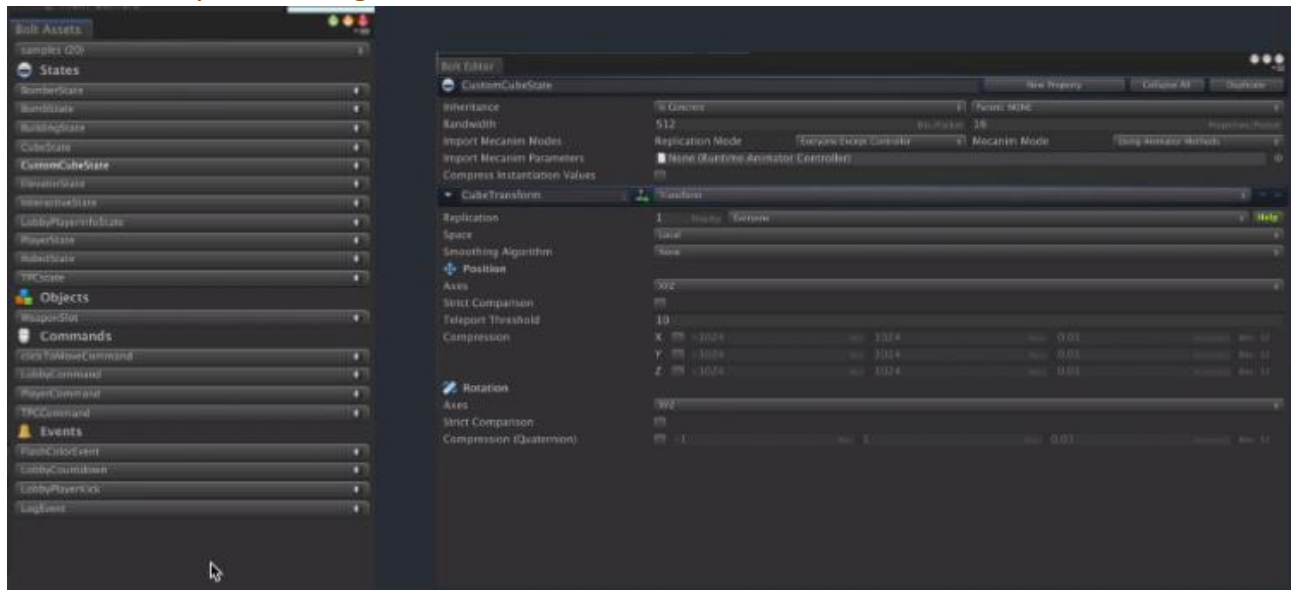
High and Low Level	PUN2	Photon Bolt	Nakama	Unity DOTS networking	Mirror	MLAPI	Forge	
General								
Networking Model (Client/Server)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Unity Integration	Yes	Yes	Yes	Yes :)	Yes	Yes	Yes	
Server Authority	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Host migration	No	Yes, Build in. (Needed)	No..?	-	Not built-in. Requires self development	Not built in	Not built-in. Requires self development	
Server Hosting (AWS, Azure)	Yes (But only windows)	Yes (But only Windows)	Yes	Yes	Yes	Yes	Yes	
Fast Messaging/Byte size	No	No	Need to Test	Messages can hold up to ~80 players	Yes	Yes	Yes	
Scoping	Yes	Yes	No	-	Yes?	Yes	Yes	
Open Source	No	No	Yes	No	Yes	Yes	Yes	
Supports 25-100 Players	No	Need to Test	Need to Test	DOTS FPS targets ~80 players	Yes, >100 easily	Need to Test	Need to Test	
CCUs	20 (free, then 100/500)	20 (free, then 100/500)	Need to Test	Unlimited	Unlimited	Unlimited	Unlimited	
Cross-Platform	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Transport Protocol	UDP	UDP	gRPC, HTTP, Websockets and rUDP	UDP/RUDP	Depends on Transport but flexible	Depends on transport but flexible	UDP/RUDP, Option for TCP?	
Dedicated Game Servers	Yes	Yes	Yes	-	Yes	Yes	Yes	
Direct IP connection (for testing)	Yes	Yes	Yes..?	-	Yes	Yes	Yes	
Server Plugins (can be made)	Yes	Yes	Offers runtime scripting with Lua	-	Yes	Yes	Yes	
Good version control			Yes	Yes	Yes	Yes	Yes	
Headless server	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Unity Runtime Server			Yes	Yes	Yes	Yes	Yes	
Serialization Handled	No	Yes	No?	No?	Yes	Yes	Yes	
Updated Frequently	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
UnPricey	No	No	Yes (But scalability is pricey)	Yes	Yes	yes	Yes	

1.2 Possible networking stacks to choose for Unity



2.1 Photon Bolt workflow examples

2.1.1 Bolt Unity Editor Integration



2.1.2 Bolt Coding style

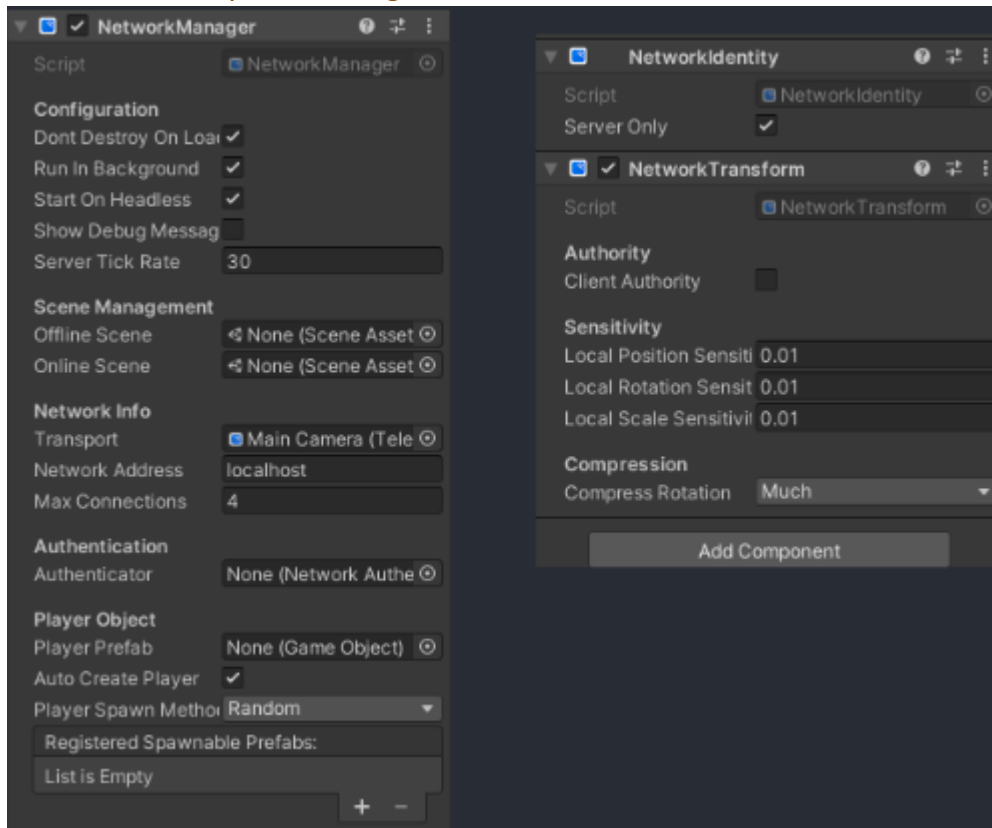
```
public class PlayerMovement : Bolt.EntityBehaviour<ICubeState>
{
    // void Start()
    public override void Attached()
    {
        state.SetTransforms(state.CubeTransform, transform);
    }

    // void Update()
    public override void SimulateOwner()
    {
        var speed = 4f;
        var movement = Vector3.zero;

        if (Input.GetKey(KeyCode.A))
        {
            movement.x -= 1f;
        }
    }
}
```

2.2 Mirror Networking workflow examples

2.2.1 Mirror Unity Editor Integration



2.2.2 Mirror Coding style

```
using UnityEngine;
using Mirror;

public class PlayerController : NetworkBehaviour
{
    [SyncVar(hook = nameof(SetColor))]
    Color playerColor = Color.black;

    // Unity makes a clone of the Material every time GetComponent<Renderer>().material is used.
    // Cache it here and Destroy it in OnDestroy to prevent a memory leak.
    Material cachedMaterial;

    public override void OnStartServer()
    {
        base.OnStartServer();
        playerColor = Random.ColorHSV(0f, 1f, 1f, 1f, 0.5f, 1f);
    }

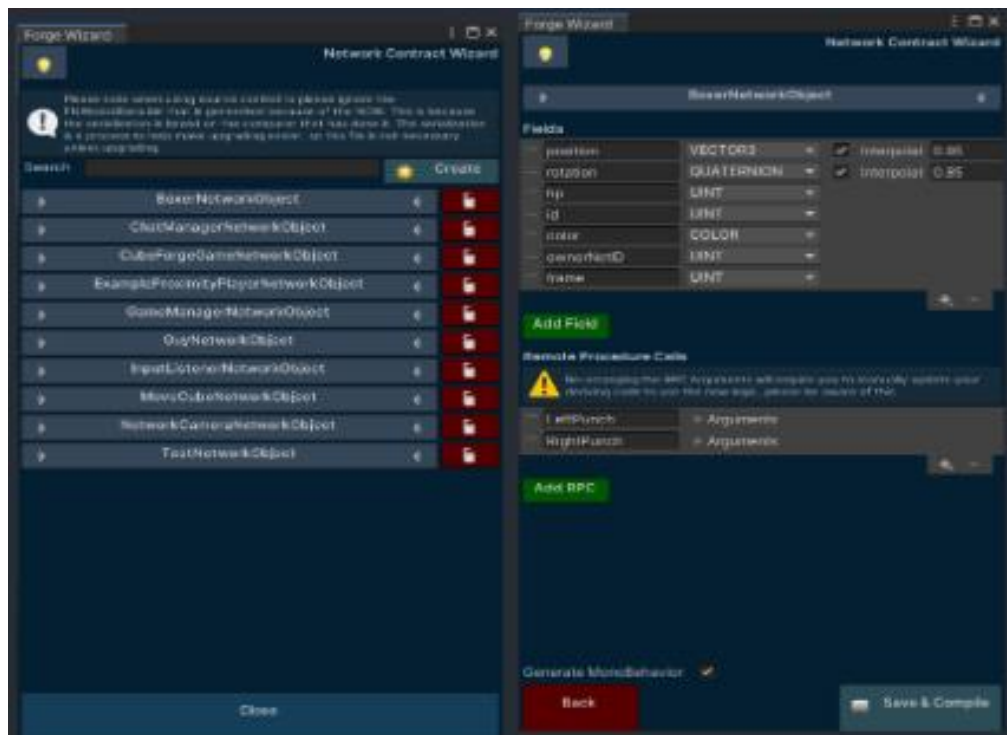
    void SetColor(Color oldColor, Color newColor)
    {
        if (cachedMaterial == null)
            cachedMaterial = GetComponent<Renderer>().material;

        cachedMaterial.color = newColor;
    }

    void OnDestroy()
    {
        Destroy(cachedMaterial);
    }
}
```


2.3 Forge Networking remastered workflow examples

2.3.1 Forge Unity Editor Integration



2.3.2 Forge Coding style

```
private void Update()
{
    if (Input.GetKeyDown(KeyCode.Space))
        networkObject.SendRpc(RPC_MOVE_UP, Receivers.AllBuffered);

    if (!networkObject.IsServer)
        return;

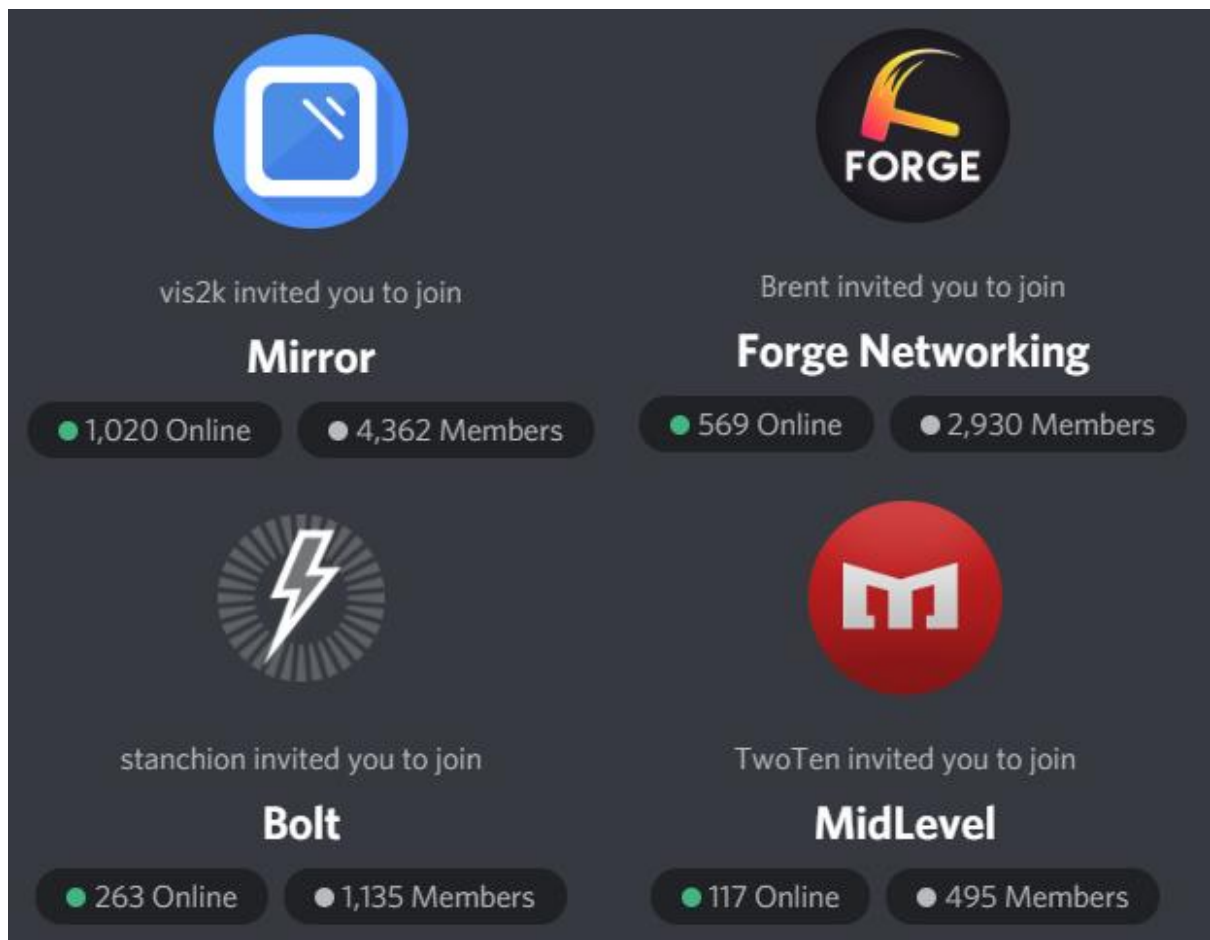
    if (Input.GetKeyDown(KeyCode.B))
    {
        Vector3 randPos = new Vector3(Random.value * 5.0f, Random.value * 5.0f, Random.value * 5.0f);
        networkObject.SendRpc(RPC_DO_SOMETHING, Receivers.All, "Forge", randPos, ++count);
    }
}

public override void MoveUp(RpcArgs args)
{
    transform.position += Vector3.up;
}

public override void DoSomething(RpcArgs args)
{
    string name = args.GetNext<string>();
    Vector3 pos = args.GetNext<Vector3>();
    int counter = args.GetNext<int>();
}
```

3.1 Discord servers compared by member count

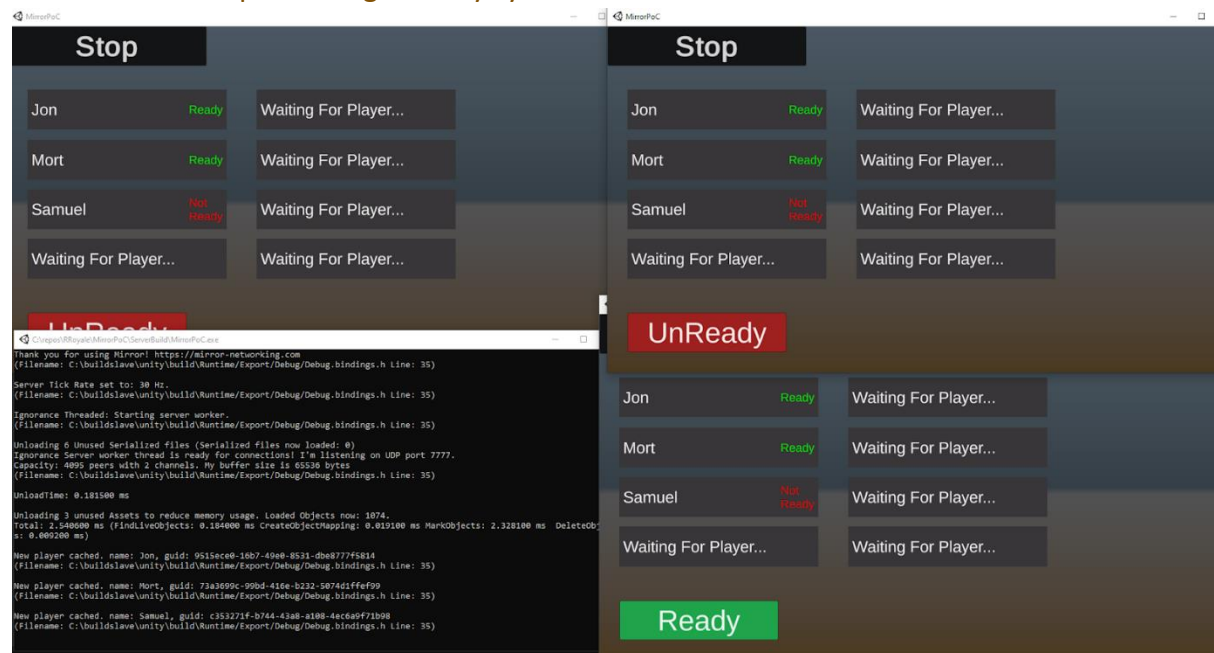
– retrieved 25th of May 2020



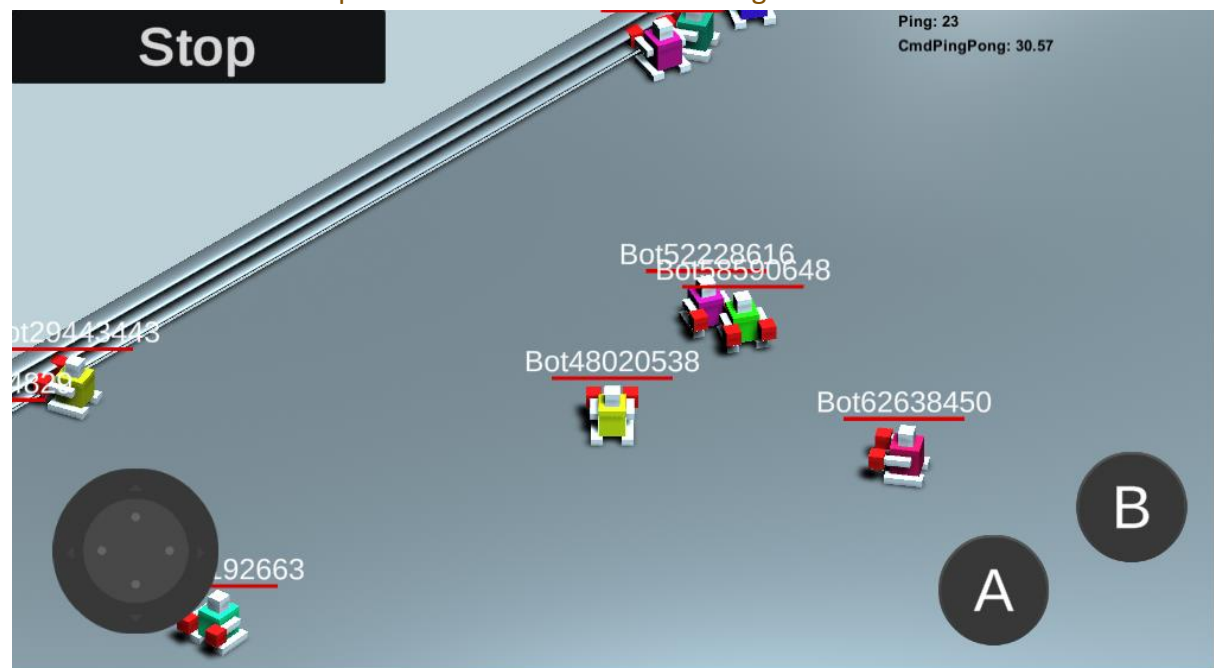
Technology	Invited By	Online	Members
Mirror	vis2k	1,020	4,362
Forge Networking	Brent	569	2,930
Bolt	stanchion	263	1,135
MidLevel	TwoTen	117	495

4. Screenshots of the prototype

4.1 Screenshot representing a Lobby system and a headless Windows sever.



4.2 AI controlled clients spawned over the network during a load test.



4.3 Server metrics after a real-world scenario test with 13 clients connected concurrently.

