ITU/JWCC

Network Optimizing Software Solution for Multiplayer Gaming

Arda Deniz Çelik[1], **Gökhan Seçinti**[1]

[1] Department of Computer Engineering, Istanbul Technical University, Istanbul, 34469, Turkey

Abstract: In the digital entertainment landscape, multiplayer online games have gained immense popularity, attracting millions globally. This paper focuses on improving gaming performance by addressing two major challenges: latency and packet loss. Utilizing the Omnet++ application with the INET framework, we propose innovative network solutions to enhance the overall gaming experience. Over the past decade, multiplayer games have rapidly gained popularity due to technological advancements. As living costs rise, these games have become an economical and accessible form of entertainment, especially among the youth. The growing importance of a stable, high-quality internet connection in gaming has become crucial. Game developers and online gaming platforms also stand to gain from our network optimization solutions. Implementing these advancements can elevate service quality, leading to increased user satisfaction, positive reviews, and a stronger reputation in the gaming community. Additionally, this paper explores using real-world gaming data to simulate and test network strategies in Omnet++, offering practical insights for enhancing real-time online gaming experiences.

Keywords: End-to-End game latency, real-time packet loss, latency-sensitive network protocol, core network QoS for gaming.

Çok Oyunculu Oyun için Ağ Optimizasyon Yazılımı Çözümü

Özet: Dijital eğlence dünyasında, çok oyunculu çevrimiçi oyunlar büyük popülerlik kazanmış ve dünya genelinde milyonları kendine çekmiştir. Bu proje, oyun performansını iyileştirmek için gecikme ve paket kaybı gibi iki ana zorluğu ele almayı hedeflemektedir. Omnet++ uygulaması ve INET çerçevesi kullanılarak, genel oyun deneyimini geliştirecek yenilikçi ağ çözümleri öneriyoruz. Son on yılda, çok oyunculu oyunlar teknolojik ilerlemeler sayesinde hızla yaygınlaşmıştır. Artan yaşam maliyetleriyle birlikte, bu oyunlar özellikle gençler arasında ekonomik ve erişilebilir bir eğlence biçimi haline gelmiştir. İstikrarlı ve yüksek kaliteli bir internet bağlantısının önemi bu süreçte daha da belirgin hale gelmiştir. Oyun geliştiricileri ve çevrimiçi oyun platformları da ağ optimizasyon çözümlerimizden faydalanmaktadır. Bu gelişmeler, hizmet kalitesini artırarak kullanıcı memnuniyeti ve olumlu geri bildirimlerle daha güçlü bir itibar kazandırabilir. Ayrıca, proje kapsamında Omnet++'da gerçek oyun verileri kullanılarak yapılan simülasyonlar, gerçek zamanlı çevrimiçi oyun deneyimlerini iyileştirmeye yönelik pratik bilgiler sunmaktadır.

Anahtar Kelimeler: Uçtan uca oyun gecikmesi, gerçek zamanlı paket kaybı, gecikmeye duyarlı ağ protokolü,oyun içi çekirdek ağ kalite hizmeti.

RESEARCH PAPER

Corresponding Author: Gökhan Seçinti, secinti@itu.edu.tr

Reference: A. D. Çelik, G. Seçinti, (2024), "Network Optimizing Software Solution for Multiplayer Gaming," *ITU Journ. Wireless Comm. Cyber.*, 1, (1) 47–56.

Submission Date: Aug, 27, 2024
Acceptance Date: Sept, 26, 2024
Online Publishing: Sept, 30, 2024

1 INTRODUCTION

It is the digital world that has revolutionized online multiplayer gaming, engaging millions of players across the globe in different platforms. With these games evolving, however, the demand for improved network performance is fast becoming an imperative. Among the most critical challenges, latency and packet loss both affect the quality of experiences; in an environment filled with competition or a league, decisions are made in a split second and matter a lot. Addressing these issues is important for improving the user experience and ensuring fair gameplay; this is particularly a need in the professional eSports context.

However, existing work has been focused on the performance of gaming and the relation between network characteristics like latency, packet loss, and player behavior. For instance, the effect of network quality on player exit behavior has been investigated by several studies, which indicate that poor network conditions may cause early player churn from games. Other works have investigated different network architectures and protocols, with a clear emphasis on genre-specific solutions that cater to the very specific demands of different classes of games. Such works represent the importance of network performance optimization for an improvement in gaming experiences across a different genre. Building on this work, our study implements and further develops network optimization approaches within the Omnet++ simulation environment, focusing on low-latency jitter handling in a multiplayer gaming context. By exploiting real gaming-world data using advanced simulation techniques, we hope to give practical answers that will not only enhance network performance but also be beneficial for the greater research into online gaming.

The major contributions of this paper are as follows:

i. A comprehensive analysis of network attributes affecting latency and packet loss in multiplayer games.

ii. Development and implementation of optimized network strategies within the Omnet++ environment.

iii. Evaluation of these strategies' impact on game performance, particularly in reducing latency and packet loss, thereby enhancing the overall gaming experience.

The rest of the paper is organized as follows: **Section II** summarizes the related work. Next, **Section III** presents the system model and methodology. **Section IV** showcases simulation setup and the performance evaluation. Lastly, **Section V** concludes the paper and discusses potential future work.

2 RELATED WORKS

In some studies which are about [1], [2], [3] FPS game traffic analysis, extensive research was conducted to investigate the influence of internet latency on game performance. Utilizing tools such as the Ethereal packet sniffer and Ns2 simulation, the study examined how various network attributes impact latency by analysing inter-send and inter-arrival times through bandwidth analysis on a clean and realistic client within the Quake III game [1]. In addition of the referenced study that primarily focused on analyzing traffic characteristics, the approach taken involves the active modification of the network environment to enhance real-time gaming experiences. This extension of research is indicative of a more practical application of network performance optimization in gaming scenarios.

Some researchers Ben, Youry and co-authors have used the TechEmpower company's 12-step test procedure to examine in detail what kind of delay data different web frameworks encounter after the benchmark. To analyze the results of these tests, they examined almost 400,000 gaming sessions using a web application called Gperf2 Collector [4]. The paper shares this focus on latency but differs in methodology. While the study relied on a virtualized network for understanding latency effects, paper uses real-world gaming data in Omnet++ simulations. This approach allows to directly apply network theory to practical gaming situations, offering more tailored solutions for improving gaming network performance.

The comprehensive analysis of Netcode in online multiplayer games, focusing on aspects such as ping, server tick rate, and the extent of delay in game dynamics for each game, resonates with the objectives of the paper. This field is advanced by the work undertaken, which not only involves analyzing network issues but also seeks to mitigate them through network optimization strategies [5]. The integration of tools such as Wireshark and Ping-Plotter, as outlined in the study, is a fundamental aspect of this paper. This highlights the practical application and relevance of these tools in addressing real-world network optimization challenges.

In the study "Client-Side Network Delay Compensation for Online Shooting Games, the authors specifically focused on synchronization and unfair gameplay in multiplayer shooting games [6]. The article suggests that current server load balancing methods are insufficient and to solve this, it recommends reducing the data used and estimating the player's location by using a regression-based method. As a result of their tests and evaluations, they claimed that the solution method they proposed provided an improvement of nearly 16 pixels in estimating player location. By using this method, it has made a significant contribution to the development and reduction of the queuing time of this paper on Omnet++.

In study "A Multiplayer Real-Time Game Protocol Architecture for Reducing Network Latency" introduces an original client/server architecture for multiplayer real-time video games to reduce network latency [7]. An STU segment queue approach serializes sporadic events into periodic groups for scheduling efficiency, and a spatial domain approach selects clients affected, updating only those clients to reduce the data transmission. The implemented architecture in Java, with non-blocking I/O and thread pool management, makes servers more efficient, reusing resources to increase the throughput of garbage collection. Our proposed architecture's experimental results show clearly reduced latency and synchronization very well for real-time gaming environments with high-quality service. These results have implications for our paper and enable approaches that reduce latency and increase synchronization and can be integrated into our system for better scaling.

Petlund study focuses on [8] the challenges of reducing latency in interactive applications, particularly online multiplayer games, that generate "thin streams." In sessions of ”thin stream” interactive applications, especially multiplayer games, traditional TCP mechanics decrease performance because packets with small payload and high interarrival are sent as a stream on the network. It is emphasized that congestion control and recovery mechanisms, which are features of TCP, are almost never triggered due to the inadequacy of the sent packets. In order to improve the TCP protocol, which is inadequate due to these packet features, the author, who focused on the game where 30-byte packets called BZFlag were sent and the packet interarrival time was 24ms, observed that the delays in the application and transport layers were significantly reduced by removing the exponential backoff and a few different modifications to reduce the delay experienced by the player. The study conducted in this paper, it has been revealed that although UDP is more difficult to create and less efficient in terms of security, it is the protocol that provides less delay in games thanks to its customizability and the simplicity of the mechanisms within it.

Many studies [2] [9] focuses on which network architecture and network design should be chosen for game genre specific because they found out that there is no superior architecture for every game genre is In Moll's study [2], they focused on the problems of network inadequacy in modern multiplayer games, especially in the Battle Royale genre. Unlike the traditional network protocols used in these games, they proposed to use Information-Centric Networking (ICN) and Named Data Networking (NDN) to develop network protocols with the data they obtained during the game sessions of nearly 36 hours in order to meet the low latency requirement in competitive games. Thanks to this proposal, they managed to significantly reduce network latency and optimize bandwidth usage. The research conducted contributed to the evaluation of possible network solutions by providing data for the test environments created in the creation of this paper.

In the study conducted by Chen [10], they tested the effect of network quality on Massively Multiplayer Online Role-Playing Games. Based on 1.356 million packet data, they observed that players left the game prematurely than their average game time due to packet loss and network delays. When the logistic regression model was applied, the study quantifies player "intolerance" to various network impairments, finding that the degrees of player intolerance to network delay, delay jitter, client packet loss, and server packet loss are approximately in the proportion of 1:2:4:3. In addition, since packet loss is the network variable that causes the most player loss, it was observed that directing players with poor internet connection to better quality servers increases the average game time overall. Thanks to this study, it played an important role in determining our main variable target as packet loss in the research conducted in this paper.

In the study [11], authors mention that they conducted a comprehensive study on the effects of network delays on the gaming experience for end users. They have shown that many different parameters such as game type, environmental factors and distance to servers are important for delays in games. For the tests carried out in this study, they created an analysis by using real-time data received through the WTFast GPN application and passing it through machine learning, which determines the game sessions at different latency levels they developed with high precision. As a result of this analysis, it was revealed that network delay is the most important factor in gaming experience and it was revealed that the machine learning model with the Random Forest algorithm was the most effective model in determining game delays among 8 different machine learning models for reducing network delay. This study has shown that machine learning can be an important supporting factor in developing gaming experience.

In the study [12], the author has found a study using game theory on dynamic resource sharing for production networks with stochastic demand. He also stated that production systems, especially those using cloud systems, are the basis for creating this simulation. With the solution presented by the author, it has been shown that, unlike traditional models, namely dedicated and full-flexibility systems, the Gale-Shapley algorithm plays a much more effective role in the trade-off between performance and the complexity caused by active links in previous studies. She also proposed that a fast computational process (scaling at $n\hat{2}$ proposals), making it ideal for real-time reconfiguration in production environments. Thanks to the solution proposed by the author, it has also been stated that it is as reliable as traditional models against demands in changing conditions and has a lower network complexity.

3 SYSTEM MODEL

3.1 Omnet++ with INET Framework

Omnet++ is a network simulation platform used to create and analyze a realistic multiplayer gaming environment. The INET framework was integrated with Omnet++ to model various network components such as hosts, routers, switches, and game servers. This setup enabled detailed simulations of network behavior under different conditions, with a focus on minimizing latency and reducing packet loss.

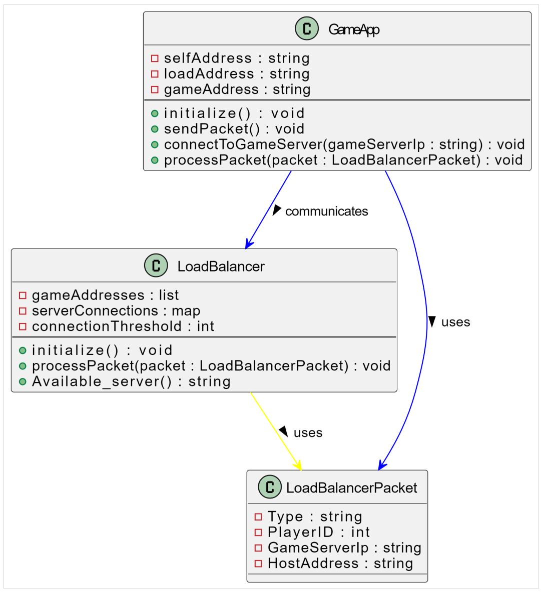

Fig. 1 Class diagram of solution.

3.1.1 Structural model for GameApp

In our multiplayer gaming simulation, **GameApp** is designed to handle interactions that are effective on the client side and with both the loadserver and the game server. This model has many important roles such as managing network connections, ensuring the effective distribution of game data, and processing the data received by the loadserver.Here's a detailed breakdown of the structural model of the **GameApp** module, covering its key algorithms and functional flows:

Initialization and Connection Setup

The initialization of the **GameApp** starts with the `processStart()` function. This function is responsible for setting up network sockets and initializing the connection settings.

```
procedure processStart
    initializeSocket()
    loadAddress <- getLoadBalancerAddress()
    connectLoadServer()
end procedure
```

This function determines the socket options for the host using GameApp to connect to the loadserver, and also ensures that the host's own address and loadserver adress to be sent to the loadserver is kept as an L3Address variable so that it can be sent. It then calls the connectLoadServer function to send the first packet and obtain the IP address of the appropriate game server.

Connecting to the Load Server

Once the initial setup is complete, the `connectLoadServer()` function is called. This function creates and sends a packet to the load balancer to determine an appropriate game server for the client to connect to, based on current server loads.

```
procedure connectLoadServer
    packet <- createPacket("Connection Request")
    sendPacket(loadAddress, packet)
end procedure
```

In this function, the first package is created and sent in order to establish a connection to the loadserver. Additionally, it is first checked whether there is any fragmentation error, and after function set the essential and custom variables to proper values to ensure communication between network layers, function send packet to loadserver using loadAdress which is stored in processStart function.

Processing Packets from the LoadBalancer

The `processPacket()` function takes over upon receiving a packet from the load balancer. This function processes different types of packets based on their content, primarily focusing on handling server assignment packets.

```
procedure processPacket(packet)
    switch packet.type
        case "Server Assignment":
            gameServerIp <- packet.getGameServerIp()
            destAddress.push_back(gameServerIp)
            processSend()
    end switch
end procedure
```

After the packet sent by the loadserver arrives, the function checks the type of the incoming packet and it is analyzed by the loadserver module that the incoming packet was sent in the correct packet type. Then, using the gameServerAdress payload sent in the package, the destination address required for the host to connect to the game is determined. Finally, the processSend function is called.

Sending Data to the Game Server

The `processSend()` function uses the inherited functionalities from `UdpBasicApp` to handle the actual data transmission to the assigned game server.

```
procedure processSend
    sendPacket(gameServerIp, createGamePacket())
end procedure
```

The ProcessSend() function creates and sends the first package whose destination address has been determined by processpacket and is ready to connect with the game server. While doing this, it uses the function inherited by UdpBasicApp.

Detailed Packet Handling

The `sendPacket()` function constructs a packet with the appropriate type and content, then sends it to the designated server address. This is crucial for game state updates and player actions.

```
procedure sendPacket(address, data)
    packet <- new Packet(data)
    packet.setType("APP_SERVER")
    network.sendTo(address, packet)
end procedure
```

This function ensures that each packet is correctly formatted and dispatched to maintain continuous and real-time gameplay interaction.

All functions in the **GameApp** application are a whole that allows the hosts to find a suitable game server through the loadserver and to process the packet received by the loadserver and to be in constant communication with the game server.

3.1.2 Structural model for LoadBalancer

The **LoadBalancer** module within our OMNeT++ simulation plays a critical role in managing server loads and distributing client requests across multiple servers. Its design ensures efficient processing of network connections and server assignments, pivotal for maintaining an optimal performance environment in multiplayer gaming scenarios. Below is a detailed explanation of the structural model of the **LoadBalancer** module, including key algorithms and pseudocode implementations.

Initialization Process

The initialization of the **LoadBalancer** leverages inheritance from the `UdpBasicApp` to set up initial module configurations.

```
procedure Initialize
    UdpBasicApp.Initialize()
    setupInitialConfiguration()
end procedure
```

This function calls the initialization method of the inherited `UdpBasicApp`, ensuring that all underlying network functionalities are correctly set up.

Starting the Process

The `processStart()` function is crucial for preparing the LoadBalancer to handle incoming connections and manage server addresses effectively.

```
procedure processStart
    bindSocket()
    for each address in initialGameAddresses
        gameAddressStr.push_back()
    end for
end procedure
```

After first determining the sockets for the loadserver, the ProcessStart() function makes preliminary adjustments to find suitable game servers by pushing the game server addresses given as parameters in the omnetpp.ini file to an L3Adress vector and defining an integer variable for each IP address.

Handling Messages

The `handleMessageWhenUp()` function utilizes the capabilities inherited from `UdpBasicApp` to manage incoming network messages.

```
procedure handleMessageWhenUp(message)
    super.processMessage(message)
end procedure
```

This function ensures that messages either from upper or lower network layers are processed according to the base app's logic.

Server Availability Check

The `Available_server()` function determines the optimal server for new connections based on current load conditions.

Iterates through the entire list of game server addresses, checking whether each function has a connection count greater or less than the connectionThreshold integer value. If there is an IP address smaller than connectionThreshold, this IP address is returned as the function result. If there is not one, the threshold value is overridden and reassigned to the servers.

Packet Processing

The `processPacket()` function is tasked with handling incoming packets from hosts, determining the available server, and responding appropriately.

```
procedure processPacket(packet)
    serverIp <- Available_server()
    responsePacket <- createPacket(serverIp)
    sendPacket(packet.origin, responsePacket)
    updateConnectionCount(serverIp, increment)
end procedure
```

The ProcessPacket() function first checks the type of the incoming package. If the package is a package sent by one of the host to loadserver, the host address is extracted from the incoming package and given to the newly created package as the destination address. In order for the payload to be sent correctly, variables are set and the L3 game server address sent by the Available_server() function is added to the package and the package is sent to the host. Finally, the connection number of the game server address sent to connect the host is updated to +1.

Connection Count Management

Lastly, the `updateConnectionCount()` function adjusts the count of current connections per server, ensuring accurate load tracking.

```
procedure updateConnectionCount(serverIp)
  if increment
      serverConnections[serverIp]++
  else
      serverConnections[serverIp]--
  end if
end procedure
```

This function modifies the connection count based on whether a new connection is being added or an existing one is terminated.

With the help of all the functions in the Loadbalancer application, game server addresses are kept through a map in the loadserver, incoming packages are processed and sent to their new destinations with the appropriate game server addresses. This is the most important of the improvements made to improve the gaming experience.

3.1.3 Structural model for LoadBalancer packet

The **LoadBalancerPacket** plays a crucial role in the communication framework of our networked multiplayer game environment, acting as the primary data carrier between the **GameApp** and **LoadBalancer** modules. This packet is designed to encapsulate all necessary data for effective load balancing and server assignment. Below, we detail the structural model of the **LoadBalancerPacket**, including its design and key functionalities.

Packet Type Enumeration

The packet types are defined by an enumeration `APP_TYPE`, which specifies the kinds of packets that can be generated within the system. This helps in distinguishing the purpose of each packet as it flows through the network.

```
enum APP_TYPE {
    CONNECTION_REQUEST,
    SERVER_ASSIGNMENT,
    DATA_TRANSFER
}
```

This enumeration facilitates the handling of packets based on their type, improving the routing and processing efficiency within the network.

Class Definition and Attributes

The class `LoadBalancerPacket` is designed with attributes that support comprehensive data encapsulation for network operations.

```
class LoadBalancerPacket {
    APP_TYPE type
    int playerID
    string gameServerIp
    string hostAddress
}
```

Here are the custom parameters of the `LoadBalancerPacket`: - `type`: An instance of `APP_TYPE`, determining the packet's function within the network. - `playerID`: An identifier for the player sending the request, aiding in response management and data tracking. - `gameServerIp`: The IP address of the assigned game server, used in server assignment operations. - `hostAddress`: The originating address of the packet, typically used for routing back responses or data.

3.2 Netlimiter

NetLimiter was employed to monitor and manage network traffic during gaming sessions. By analyzing download and upload rates for each application, NetLimiter provided essential insights into how the network behaves in real-time. The IP filtering feature allowed the study to focus on specific network interactions, particularly those related to the game servers, enabling a targeted approach to traffic analysis.

3.3 Wireshark

Wireshark, a network protocol analyzer, was used to capture and examine packet data flowing through the network during gaming sessions. It provided detailed information on packet sizes, transmission intervals, and overall traffic patterns. This data was instrumental in refining the simulation model within Omnet++, ensuring that it accurately reflected real-world network conditions. Wireshark's ability to filter and analyze traffic based on specific IP addresses, identified through NetLimiter, made it a vital tool in the study's methodology.

4 PERFORMANCE EVALUATION

The network design where the main tests of the paper are carried out is seen in the Figure. This figure consists of both 2 different game servers and hosts that want to play more than one game at the same time to create and test a realistic game environment. In order to connect to the game servers, the hosts first pass through their local switches to the routers and then through the ISP router to reach the servers. The important aspect of the design is that instead of the hosts connecting directly to the servers, a load balancer server is created that decides which servers are more suitable for the hosts to connect to. In this network design, thanks to the customized packages in the load balancer, the connection availability statuses determined by the instant occupancy rates of the servers are kept in the load balancer server and it allows the host to direct the incoming game connection request to the most suitable game server for itself. Additionally load balancer made the best game server selection is decided for each user in a way that allows players to benefit from less packet loss and delay.

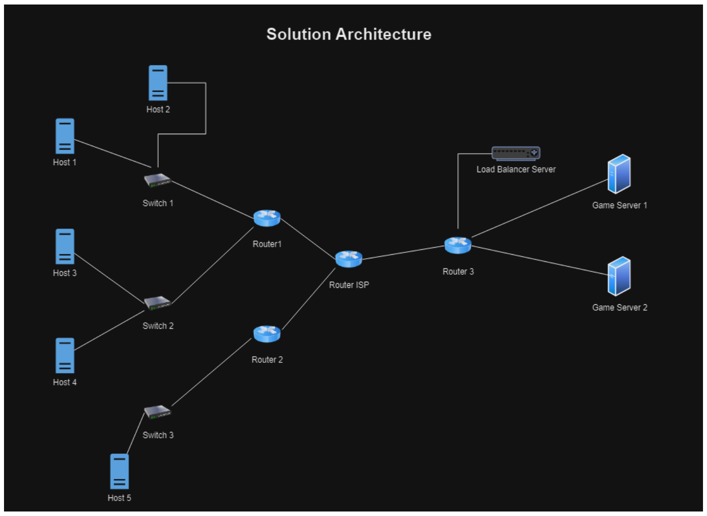

Fig. 2 Solution design in Omnet++.

In the study carried out using Ethereal packet sniffer and Ns2 simulation on behalf of FPS games, [1] focused only on the Quake III game, examined the network features and traffic structure of this game, and carried out studies to improve the gaming experience by making modifications, especially in terms of throughput and delay. In this study, the game traffic of more than one game was examined using NetLimiter and WireShark applications to ensure less delay and throughput. As a result of work have done to prevent queuing time and possible delays that may occur, even if only in small numbers,the packet loss was successfully reduced to a value very close to zero, effectively preventing potential instant delays, as demonstrated in Figure 3. To achieve this by adding a slight delay and allowing each game server to send packets one by one, instead of constantly bombarding the game servers with packets at the same time. This enabled to achieve the goal of processing 1000 operations per second that.

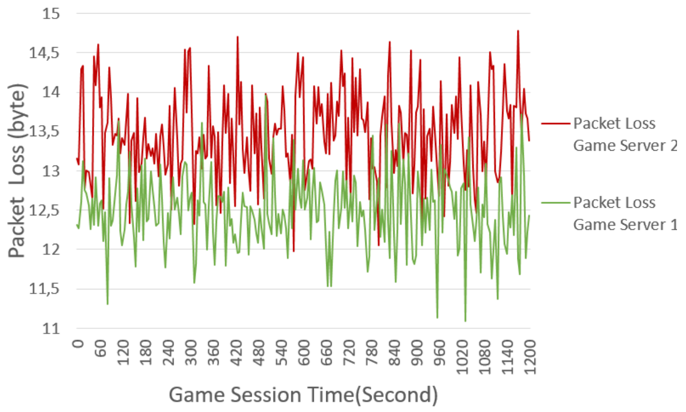

Fig. 3 Packet loss of game servers.

In [4] study, an average of 400,000 gaming sessions are examined using TechPower company procedures. The web application called Gperf2, which they tested in this study, inspired this paper to develop the queuing time of the model. By using the approach of this team, which also tested the problems that may be caused by overloading the server with the techniques they used, the queuing time value shown in Figure 4 is reached for game server 1 and game server 2, which shows significant improvement with the use of Load balancer rather then without using the Load balancer module. In this way, queuing time development is completed to reduce instant spikes in queueing time. Use of Load balancer was increased the sharpness of the sended packet response time and improve the overall quality of the game for players. Additionally, with the reduced queueing time for servers, overall player play time will be increase significantly due to more reliable conenction status.

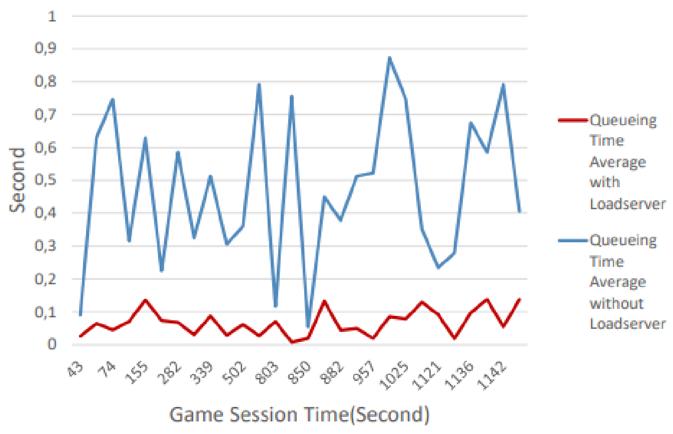

Fig. 4 Loadserver queueing time.

The goal of reducing the observed delay between 5% and 7% and packet lose by 8%, the research paper [5] inspired this work to achieve the goal. The main purpose of this research, which makes a comparative Netcode analysis of online multiplayer games, is to examine the essential data of the game experience such as ping and server tick rate and find solutions to the problems that occur. Using applications such as WireShark and Ping-Plotter to find these solutions, the research team developed different implementation strategies for servers and included them in their research to solve the delays and packet losses that occur. In this study, thanks to the LoadBalancer and GameApp applications, both the delays that may occur on the servers and the packet losses were improved by 6.35% and overall delay reduce by 7%. In other Figure 5 represent a essential data for server. Figure give a insight about upcoming packages are passed to servers without noticable losses which is a big problem in the start phase of the paper.

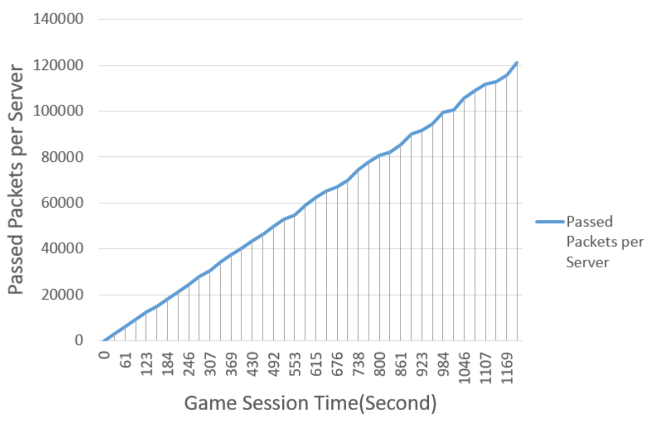

Fig. 5 Passed packet per server.

5 CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this paper, we developed and evaluated a network optimization solution for multiplayer gaming environments using the OMNeT++ simulation platform. The system demonstrated improved performance, achieving a throughput exceeding 1000 operations per second, reducing latency by approximately 7%, and decreasing packet loss rates by 6.35%. These improvements contribute to a more stable and enjoyable gaming experience.

5.2 Future Work

Future research can focus on several areas to further enhance the system:

- **Scalability Improvements**: Optimizing the LoadBalancer algorithm to efficiently manage increased traffic using more advanced algorithms.
- **Machine Learning for Load Balancing**: Implementing machine learning algorithms to dynamically adjust server allocations based on real-time network conditions.
- **Enhanced Security Measures**: Integrating advanced security protocols to protect against potential threats such as DDoS attacks.
- **Server Distribution Algorithm**: Adapting the system for more flexible and advanced server Ip adress distribution.
- **Real-Time Analytics Dashboard**: Developing a dashboard to visualize key network performance metrics for proactive management.

ACKNOWLEDGEMENTS

This work is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) 1515 Frontier R&D Laboratories Support Program for BTS Advanced AI Hub: BTS Autonomous Networks and Data Innovation Lab. Project 5239903 and Istanbul Technical University, Department of Scientific Research Projects (MÇAP-2022-43828).

REFERENCES

[1] Q. Zhou, C. J. Miller, and V. Bassilious, “First person shooter multiplayer game traffic analysis,” in *Proceedings of the 2008 International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, 2008. DOI: 10.1109/isorc.2008.28.

[2] P. Moll, M. Lux, S. Theuermann, and H. Hellwagner, “A network traffic and player movement model to improve networking for competitive online games,” in *Proceedings of the 2018 16th Annual Workshop on Network and Systems Support for Games (NetGames)*, Jun. 2018. DOI: 10.1109/netgames.2018.8463390. [Online]. Available: https://doi.org/10.1109/netgames.2018.8463390.

[3] T. Henderson, “Latency and user behaviour on a multiplayer game server,” in *Lecture Notes in Computer Science*, 2001, pp. 1–13. DOI: 10.1007/3-540-45546-9_1. [Online]. Available: https://doi.org/10.1007/3-540-45546-9_1.

[4] B. Ward, Y. Khmelevsky, G. Hains, R. Bartlett, A. Needham, and T. Sutherland, “Gaming network delays investigation and collection of very large-scale data sets,” in *2017 Annual IEEE International Systems Conference (SysCon)*, Apr. 2017. DOI: 10.1109/syscon.2017.7934779.

[5] M. Ahmed, S. Reno, M. R. Rahman, and S. H. Rifat, “Analysis of netcode, latency, and packet-loss in online multiplayer games,” in *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, Nov. 2022. DOI: 10.1109/icaiss55157.2022.10010926.

[6] T. Motoo, J. Kawasaki, T. Fujihashi, S. Saruwatari, and T. Watanabe, “Client-side network delay compensation for online shooting games,” *IEEE Access*, vol. 9, pp. 125 678–125 690, Jan. 2021. DOI: 10.1109/access.2021.3111180.

[7] Y. Ahn, A. K. Cheng, J. Baek, and P. Fisher, “A multiplayer real-time game protocol architecture for reducing network latency,” *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 1883–1889, Nov. 2009. DOI: 10.1109/tce.2009.5373746.

[8] A. Petlund, K. Evensen, P. Halvorsen, and C. Griwodz, “Improving application layer latency for reliable thin-stream game traffic,” in *Proceedings of the 7th ACM International Conference on Multimedia*, 2008. DOI: 10.1145/1517494.1517513. [Online]. Available: https://doi.org/10.1145/1517494.1517513.

[9] X. Che and B. Ip, “Packet-level traffic analysis of online games from the genre characteristics perspective,” *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 240–252, Jan. 2012. DOI: 10.1016/j.jnca.2011.08.005. [Online]. Available: https://doi.org/10.1016/j.jnca.2011.08.005.

[10] N. K.-T. Chen, P. Huang, and N. C.-L. Lei, “Effect of network quality on player departure behavior in online games,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 5, pp. 593–606, May 2009. DOI: 10.1109/tpds.2008.148. [Online]. Available: https://doi.org/10.1109/tpds.2008.148.

[11] A. Wong, C. Chiu, G. Hains, J. Behnke, Y. Khmelevsky, and T. Sutherland, “Network latency classification for computer games,” in *2021 IEEE Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2021. DOI: 10.1109/rasse53195.2021.9686848. [Online]. Available: https://doi.org/10.1109/rasse53195.2021.9686848.

[12] P. Renna, “Capacity and resource allocation in flexible production networks by a game theory model,” *The International Journal of Advanced Manufacturing Technology*, vol. 120, no. 7–8, pp. 4835–4848, Mar. 2022. DOI: 10.1007/s00170-022-09061-y. [Online]. Available: https://doi.org/10.1007/s00170-022-09061-y.