

Peer-To-Peer Implementation of The Game Client-Server Using TCP Hole Punching

Mohammad Alifian¹, Rizky Yuniar Hakkun^{1*}, Kholid Fathoni¹

¹*Game Technology Department, Politeknik Elektronika Negeri Surabaya, Indonesia*

**Corresponding author: Rizky Yuniar Hakkun; email: rizky@pens.ac.id*

Manuscript received June 20, 2026, revised August 31, 2026, accepted September 3, 2026

Abstract – *The rapid growth of online multiplayer games highlights the critical need for low-latency networks and resource-efficient server architectures, particularly for latency-sensitive genres like shooter games. Traditional Client-Server models often suffer from high network latency and require significant server resource allocation to handle continuous data transmission. To address these issues, this study implements and evaluates a parallel TCP Hole Punching technique to establish a direct Peer-to-Peer (P2P) connection within a 2D Top-Down Third-Person Shooter game environment. We compared the performance of this P2P architecture against a conventional Server-Client network. The experimental results demonstrate that the TCP Hole Punching method significantly reduces the server load, decreasing average CPU usage by 80.67% and slightly lowering RAM consumption by 8.77%. Furthermore, the data transmission speed improved significantly, with the Round-Trip Time (RTT) reduced by 69.89% (from 279ms to 84ms). Despite these performance advantages, implementing TCP Hole Punching on a larger scale remains challenging due to deployment limitations in restrictive network environments, such as asymmetric NAT configurations.*

Keywords: *TCP Hole Punching, Peer-to-Peer, Multiplayer Game, Network Latency, Network Address Translation, Server Performance*

I. Introduction

The video game industry has expanded into a massive entertainment sector, with major studios like Activision Blizzard reporting substantial annual revenues [1]. This rapid growth is largely driven by advancements in communication technologies, particularly the widespread adoption of 4G and 5G networks [2], which has significantly increased global internet accessibility including in Indonesia, where internet penetration has reached 73.7% [3].

However, unlike single-player titles, multiplayer games are strictly dependent on stable internet connectivity. Network latency, or "lag," profoundly impacts player performance, especially in interactive, fast-paced genres [4]. Research indicates that first-person or third-person shooter games are the least tolerant of network delays because players directly and immediately control character movements, whereas real-time strategy games can tolerate slight delays due to their indirect control mechanics [5].

Traditionally, multiplayer architectures rely on centralized Client-Server models to handle interactions. While reliable, this model often introduces higher latency and requires substantial server resources. A direct Peer-to-Peer (P2P) connection between players could solve this latency issue, but it faces a major infrastructural hurdle:

Network Address Translation (NAT). Due to the exhaustion of IPv4 addresses, NAT is widely used to allocate temporary public endpoints for private networks. Consequently, NAT typically blocks incoming traffic from unrecognized sources, preventing two nodes on different private networks from communicating directly.

To bypass this restriction, "hole punching" techniques specifically TCP Hole Punching [6] can be employed to establish a direct P2P connection between two devices behind NATs with the initial help of a rendezvous server [7].

To address the limitations of traditional network setups, this research evaluates the efficiency of parallel TCP Hole Punching in a latency-sensitive environment. The main contributions of this paper are:

1. Implementing a parallel TCP Hole Punching architecture within a custom-built 2D Top-Down Third-Person Shooter game ("King Deathmatch") on a Windows platform.
2. Providing a direct, quantitative comparison of CPU load, RAM usage, and Round-Trip Time (RTT) between the TCP Hole Punching network and a conventional Server-Client network.

II. State of the Art

A. TCP Hole Punching Approach to Address Devices in Mobile Networks

The test results in this study revealed that some hosts were unable to establish proper connections due to various internal and external factors. The primary external factor identified was the differing behaviour of NAT techniques used by mobile operators. To better understand the NAT box behaviour, a simple application was implemented to retrieve a device's own public socket details. By sending messages to different servers, the application received replies containing the public socket details assigned by the respective NAT for each communication. The results showed that the implementation worked well on the TELE2 mobile network, which assigned public IP addresses to devices without applying NAT translation. This allowed devices from any mobile network, even those with NAT, to directly initiate sessions with devices on the TELE2 network.

On the other hand, EMT mobile assigned private IP addresses to clients and translated them into public IP addresses, when necessary, while keeping the local port number unchanged. The current implementation only supported full-cone NAT, which meant that connections couldn't be initiated to the EMT network if they were using a different NAT technique. ELISA mobile network's address translation was like symmetric NAT, where a local socket was translated into a new public socket even for requests coming from the same local socket but to different destinations. This made it challenging to implement hole-punching due to the involvement of port prediction. Additionally, some NAT boxes checked the sequence numbers of incoming TCP packets, closing the public socket if the sequence number was invalid. To overcome this, capturing the sequence numbers of endpoints' TCP processes was necessary, but it required the use of raw sockets, which was difficult on Android devices without rooting them. Although a solution based on this approach was developed, it was deemed too invasive for mobile devices or IoT devices. Hence, this limitation was acknowledged as part of the implemented solution rather than the proposed solution [8].

B. NAT Hole Punching Revisited

The paper compares parallel hole punching and sequential hole punching techniques. It concludes that parallel hole punching is more effective and capable of handling non-standard and adverse conditions, while sequential hole punching is more vulnerable under the same circumstances. In terms of performance, parallel hole punching outperforms sequential hole punching due to the inherent timeout in the latter. However, sequential hole punching has an advantage in terms of implementation complexity. The paper presents Java

implementations of both techniques and highlights the superiority of parallel hole punching in most aspects. An open-source parallel hole punching implementation is provided, and future plans include integrating port prediction, developing an integrated NAT traversal approach, and exploring the use of parallel hole punching on mobile platforms. Initial results on Android support the hypothesis that modern mobile devices can handle the requirements of parallel hole punching [9].

C. SYN-TCP hole punching based on SYN injection

The researchers conducted an evaluation of their approach using a test setup consisting of 12 NAT routers. They had 10 embedded devices and 2 routers running on PC hardware. The WAN ports of the routers were connected to a Linux PPPoE server that simulated an ISP and provided routing between the devices. A UDP rendezvous server served as a communication channel. The LAN ports of the routers were connected to two test machines, one acting as a connection initiator (host A) and the other as a listener (host B). The routers were reset to factory defaults, and the TTL value of forwarded packets was manipulated using Linux Net filter on the PPPoE server to ensure they would time out in transit. The researchers implemented an automated test software using .NET C# and WinPcap. They encountered difficulties determining the external IP endpoints with three routers and excluded them from further testing. They conducted a total of 72 test cases with the remaining routers, and 56 cases (78%) were successful, while 16 cases timed out and were unsuccessful. Analysis of network traces confirmed the expected behaviour in the successful tests. The failed tests involved two routers, IPCop and the Level One FBR-1415TX, which did not meet the expected behaviour. However, if both hosts were not behind these routers, a connection could still be established by switching the roles of the initiator and listener.

The researchers discovered special characteristics of certain routers that allowed for other hole punching attempts. For example, the DigitusDN-11004-O router allowed the sequence SYN out, RST in, SYN in, enabling a TCP connection. They suggested that new hole punching mechanisms could be developed based on these characteristics. Additionally, the researchers conducted tests between hosts connected to different ISPs and found successful SYN connections in both directions, with the necessity of the SYN message varying depending on the router being used as the listener [10].

D. A Comparative Analysis of NAT Hole Punching

The 'virtual Internet' environment created for this study shows that parallel hole punching connection need times around 1 second for peer-to-peer connection to be created MacOS X. For other configurations, the mean connection

setup times range from 250ms to 690ms. In the 'real Internet' environment, sequential hole punching has a higher setup time of about 2 seconds due to the specified timeout, which may not be sufficient for certain scenarios like mobile Internet access. Reducing the timeout would require introducing retransmits, making the protocol like parallel hole punching in terms of performance and complexity. Sequential hole punching is easier to implement but requires additional mechanisms for failure recovery. Parallel hole punching requires more resources on peers and higher kernel resources for the mediator but offers better effectiveness and performance. Overall, parallel hole punching is more effective, resilient, and performs better than sequential hole punching, except in terms of implementation complexity. Resource consumption does not have a clear winner between the two approaches [11].

Based on the aforementioned studies, while various hole-punching mechanisms have been explored, there is still a need to evaluate its parallel implementation specifically within the constraints of a fast-paced 2D multiplayer game, which is the focus of this study.

III. Research Methodology

This research focuses on evaluating the server load, specifically CPU and RAM consumption, and the data transmission speed between clients, measured by Round-Trip Time (RTT). The evaluation was divided into two phases: testing the game utilizing a TCP Hole Punching connection and testing the identical game using a traditional Server-Client configuration

A. Network Structure Design

Successful hole punching requires consistent endpoint translation and, in multi-layered NAT scenarios, hairpin translation. Since securing constant private and public endpoints in a real-world scenario is challenging, a controlled network structure was established (see Fig. 1). In this configuration, one client was placed behind a constant NAT, while the second client was connected directly to the public internet. Initially, both clients connect to a central server to exchange endpoints. Once the TCP Hole Punching process is successful, the direct P2P connection is established, rendering the continuous server mediation obsolete (as illustrated in Fig. 2). The step-by-step procedure for initiating this direct connection is detailed in the process flowchart (Fig. 3).

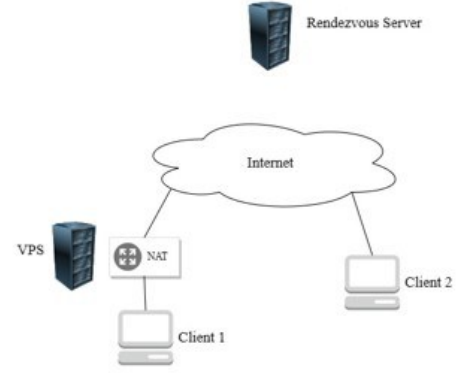


Fig. 1. Main network structure

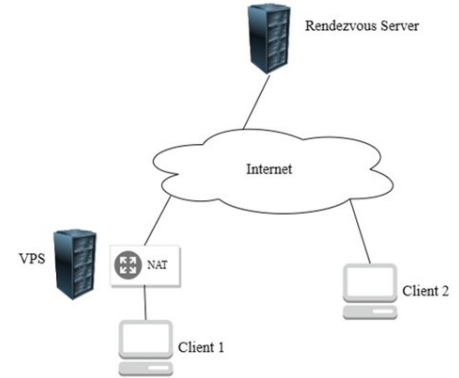


Fig. 2. TCP hole punching network structure

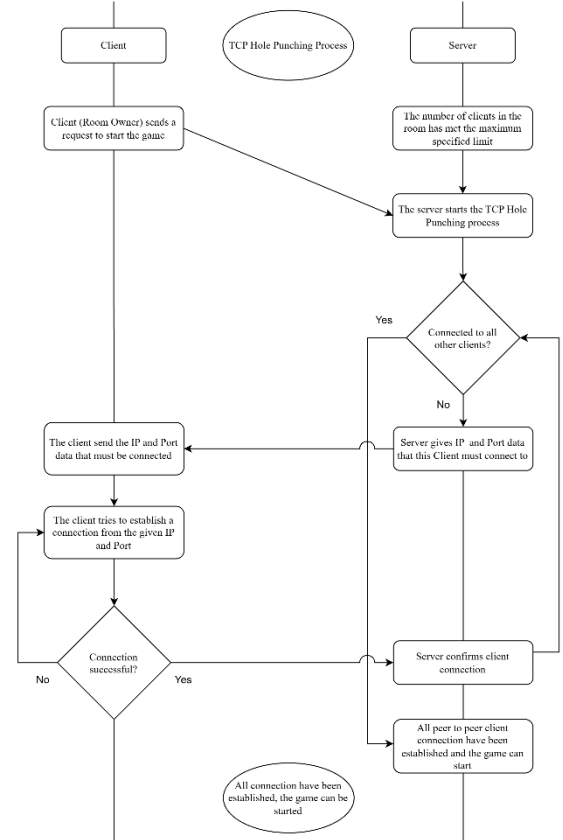


Fig. 3. TCP hole punching process flowchart

B. Game Mechanics

To test the network architecture in a latency-sensitive scenario, a 2D Top-Down Third-Person Shooter game titled "King Deathmatch" (inspired by Brawl Stars) was developed. Two identical versions of the game were created to isolate the network connection variable: one operating on a Server-Client model and the other on a TCP Hole Punching architecture. The core objective of the game is to acquire and hold the "Crown," which spawns in the center of the arena. Players acquire the crown by navigating their characters to collide or interact with it. Upon a player's death, the crown is dropped and the player respawns at the arena's edge after a brief delay. During gameplay, the server application actively logs the RTT, CPU usage, and RAM consumption into a database for comparative analysis.

C. Server Authentication and Matchmaking

The game utilizes a native C# server operating on the TCP protocol. To ensure security, a verification process is executed at the start of the connection, utilizing both asymmetric and symmetric encryption. Upon a successful three-way TCP handshake, the server and client exchange cryptographic keys. Only clients that successfully decrypt the server's challenge and establish a symmetric key are permitted to proceed (Fig. 4).

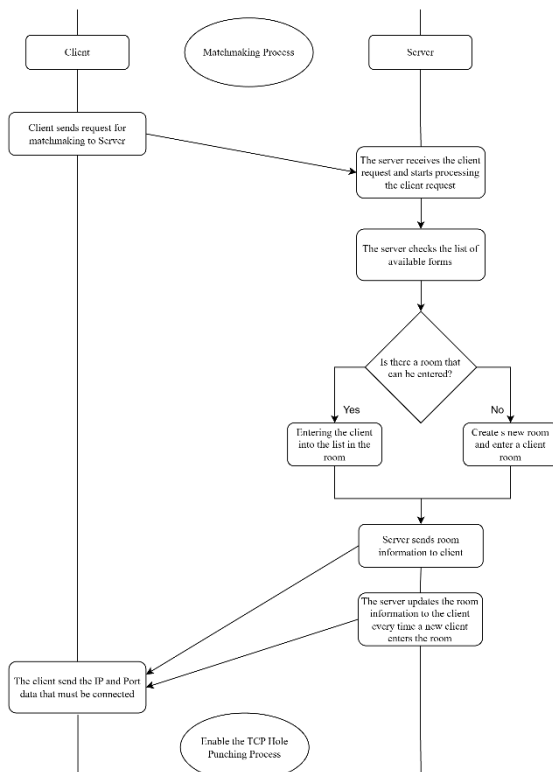


Fig. 4. Player verification process flow

Following authentication, the matchmaking process begins. The server verifies available public rooms. If a room is below its maximum capacity, the player is admitted; otherwise, the server generates a new room,

designating the player as the "Room Owner". Real-time updates regarding room ID, room name, and participant lists are subsequently transmitted to the connected players (Fig. 5).

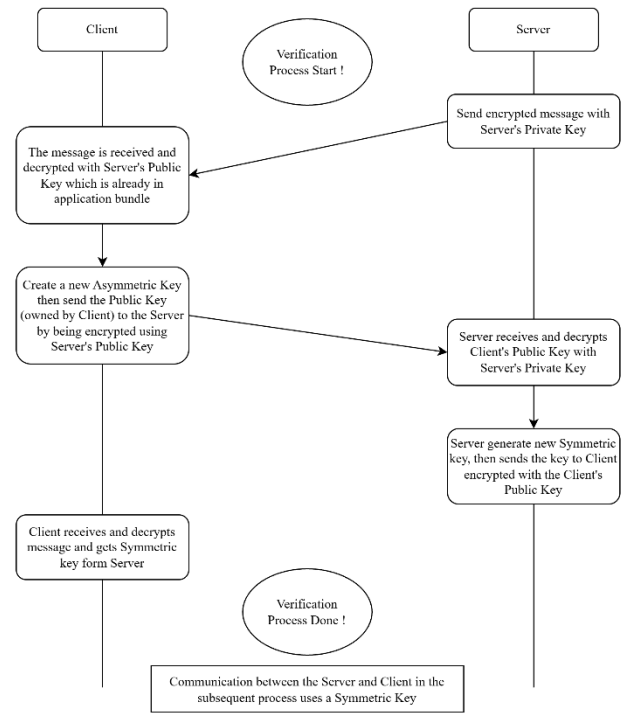


Fig. 5. Matchmaking flow

IV. Result and Discussion

To evaluate the performance of both network architectures, the game was tested over twenty iterations for each setup. The collected data (comprising CPU usage, RAM consumption, and Round-Trip Time (RTT)) are presented in Table I (TCP Hole Punching) and Table II (Server-Client)

TABLE I
HOLE PUNCHING RESULT EXPERIMENT

No	CPU (%)	RAM (%)	RTT (Ms)
1	0.09	1.31	111
2	0.01	1.51	109
3	0.03	1.64	104
4	0.1	1.27	116
5	0.03	1.5	106
6	0.11	1.27	112
7	0.02	1.46	109
8	0.02	1.59	104
9	0.03	1.59	107
10	0.02	1.58	107
11	0.12	1.29	66
12	0.06	1.31	38
13	0	1.54	59
14	0.02	1.63	38
15	0.1	1.28	77
16	0.01	1.48	64
17	0.09	1.28	56

No	CPU (%)	RAM (%)	RTT (Ms)
18	0.02	1.44	63
19	0.02	1.57	72
20	0.03	1.61	60
Average	0.047	1.458	84

TABLE II
SERVER CLIENT RESULT EXPERIMENT

No	CPU (%)	RAM (%)	RTT (Ms)
1	0.4	1.42	206
2	0.21	1.62	205
3	0.32	1.61	189
4	0.18	1.64	200
5	0.23	1.65	187
6	0.43	1.36	190
7	0.28	1.55	195
8	0.27	1.6	194
9	0.22	1.63	202
10	0.25	1.63	194
11	0.23	1.41	560
12	0.27	1.36	316
13	0.22	1.57	296
14	0.18	1.61	334
15	0.23	1.65	355
16	0.12	1.66	342
17	0.16	1.72	341
18	0.26	1.72	371
19	0.13	1.76	360
20	0.17	1.77	336
Average	0.238	1.597	279

The experimental results demonstrate a significant reduction in server load when utilizing the TCP Hole Punching method. As shown in the data, the average CPU usage for the TCP Hole Punching network is 0.047%, compared to 0.238% for the Server-Client network. This represents an 80.67% decrease (an absolute difference of 0.192%) in CPU load, highlighting the computational efficiency of bypassing continuous server mediation. (Fig. 6)

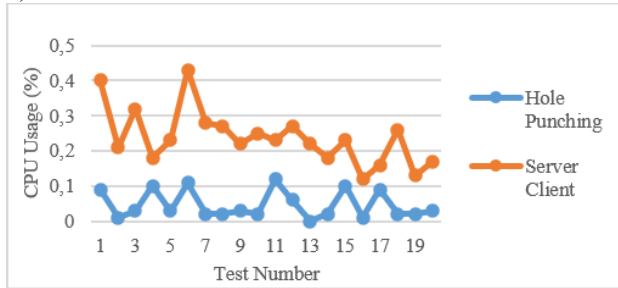


Fig. 6. Average CPU usage

Regarding memory consumption, the TCP Hole Punching architecture recorded an average RAM usage of 1.458%, slightly lower than the 1.597% observed in the Server-Client model. This yields a modest reduction of 8.77% (a difference of 0.140%, or approximately 6MB). While the impact on RAM is less drastic than on the CPU, it still contributes to overall server efficiency (Fig. 7).

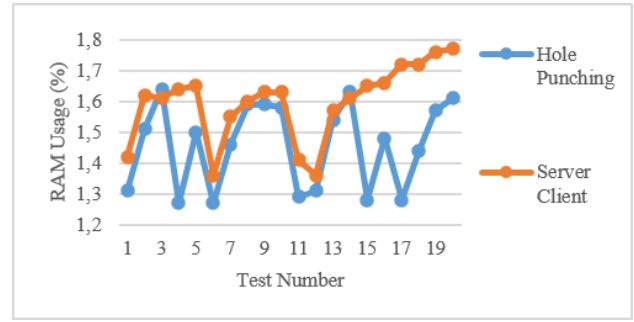


Fig. 7. Average RAM Usage

The most substantial improvement was observed in network latency. The direct Peer-to-Peer connection achieved an average RTT of 84ms, whereas the traditional Server-Client configuration averaged 279ms (Fig. 8). This 195ms difference indicates that data transmission in the P2P setup is 69.89% faster. For fast-paced, latency-sensitive games, this drastic reduction in RTT directly translates to a much smoother and more responsive player experience.

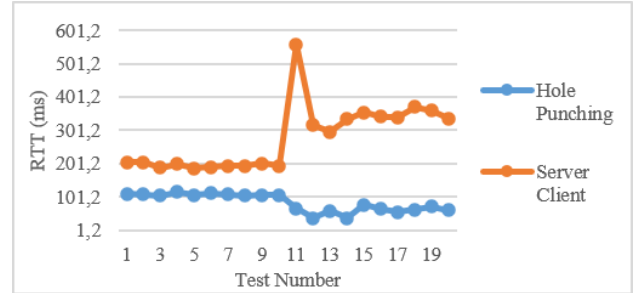


Fig. 8. Average Round Trip Time

V. Conclusion

This study confirms that implementing TCP Hole Punching for multiplayer game networking significantly optimizes server performance and reduces latency. Compared to a traditional Server-Client model, establishing a direct peer-to-peer connection reduced average CPU usage by 80.67% and improved the Round-Trip Time (RTT) by 69.89%. This provides a much faster data transmission rate, which is strictly essential for real-time interactive gaming. RAM consumption also saw a marginal improvement of 8.77%.

Despite these performance advantages, widespread implementation of TCP Hole Punching remains challenging due to strict infrastructural constraints. The primary limitation arises when clients are located behind symmetric NATs. Because symmetric NATs allocate a new public socket for every connection, making port prediction highly difficult, the direct peer-to-peer connection is prone to failure. This fundamental technical limitation is the primary reason why TCP Hole Punching is difficult to scale globally across diverse NAT infrastructures.

To address these deficiencies, future research should focus on developing more resilient hole-punching

algorithms or integrating robust fallback mechanisms. Furthermore, migrating this architecture to widely used game engines rather than relying solely on the current C# and .NET 6.0 framework would enhance the accessibility of this P2P method as an open-source networking solution for developers. Finally, conducting detailed investigations into the success rates of TCP Hole Punching across various Internet Service Providers (ISPs) and specific network types (such as mobile operators or public Wi-Fi) is highly recommended to comprehensively map its real-world viability.

References

- [1]. A. Gupta, S. Nicovich, and T. Garrison, "Electronic Arts: gambling on the future of video games," *Emerald Emerging Markets Case Studies*, pp. 1–20, Jun. 2016.
- [2]. T. Gu, "The Evolution of Gaming Through 5G," *Newzoo*, March. 9, 2020.
- [3]. APJII, "Survey Pengguna Internet APJII 2019-Q2 2020 Ada Kenaikan 25,5 Juta Pengguna Baru di RI," *Buletin APJII, Edisi 74*, Nov. 2020.
- [4]. N.-C. Wang and K.-T. Chen, "On the battle between lag and online gamers," in *Communications Quality and Reliability (CQR)*, 2011 IEEE International Workshop Technical Committee, 2011.
- [5]. W. Lamotte, A. Beznosyk, P. Quax, and K. Coninx, "Influence of Network Delay and Jitter on Cooperation in Multiplayer Games," in *VRCAI '11: Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, 2011, pp. 351–354.
- [6]. Javvin, *Network Protocols Handbook*, 2nd ed. Javvin Technologies, 2004, p. 27.
- [7]. Ford, P. Srisuresh, and D. Kagal, "Peer-to-Peer Communication Across Network Address Translators," 2015.
- [8]. S. Narayana and M. Liyanage, "TCP Hole Punching Approach to Address Devices in Mobile Networks," in *2014 International Conference on Future Internet of Things and Cloud*, 2014.
- [9]. Maier, O. Haase, J. Wasch, and M. Waldvogel, "NAT Hole Punching Revisited," in *IEEE 36th Conference on Local Computer Networks (LCN 2011)*, Bonn, Germany, Oct. 2011.
- [10]. S. Holzapfel, M. Wander, A. Wacker, and T. Weis, "SYNI - TCP hole punching based on SYN injection," in *Proceedings of The Tenth IEEE International Symposium on Networking Computing and Applications (NCA 2011)*, Cambridge, MA, USA, Aug. 2011.
- [11]. Maier, O. Haase, J. Wasch, and M. Waldvogel, "A Comparative Analysis of NAT Hole Punching," 2011.