JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 28 No. 2, pp. 39-46, February 2023
https://doi.org/10.9708/jksci.2023.28.02.039

Adaptive Packet Transmission Interval for Massively Multiplayer Online First-Person Shooter Games

Seungmuk Oh*, Yoonsik Shim*

*Student, Dept. of Game Engineering, Pai Chai University, Daejeon, Korea
*Professor, Dept. of Game Engineering, Pai Chai University, Daejeon, Korea

[Abstract]

We present an efficient packet transmission strategy for massively multiplayer online first-person shooter (MMOFPS) games using movement-adaptive packet transmission interval. The player motion in FPS games shows a wide spectrum of movement variability both in speed and orientation, where there is room for reducing the number of packets to be transmitted to the server depending on the predictability of the character's movement. In this work, the degree of variability (nonlinearity) of the player movements is measured at every packet transmission to calculate the next transmission time, which implements the adaptive transmission frequency according to the amount of movement change. Server-side prediction with a few auxiliary heuristics is performed in concert with the incoming packets to ensure reliability for synchronizing the connected clients. The comparison of our method with the previous fixed-interval transmission scheme is presented by demonstrating them using a test game environment.

▸ **Key words**: Client-server system, Server-side simulation, Packet transmission, Network game, Synchronization

[요 약]

본 논문은 클라이언트-서버 방식을 사용하는 대규모 1인칭 온라인 슈터 게임(MMOFPS)에서의 네트워크 부하를 줄이기 위한 효율적인 적응적 패킷전송 주기 방법을 제안한다. 플레이어 움직임에 있어서 빠르고 지속적인 변화와 정적이고 선형적인 상태가 다양하게 공존하는 FPS 게임의 특성상 변화의 정도에 따라 서버로의 패킷 전송량을 절약할 수 있는 지점들이 존재하는데, 이를 위해 본 논문에서는 클라이언트가 매 패킷을 전송할 때마다 플레이어의 위치 및 움직임 변수들의 변화량을 측정하여 이를 기반으로 다음번 패킷이 전송되어야 할 시간 간격을 계산한다. 서버 측에서는 받은 패킷의 정보들을 사용하여 다음 패킷이 도착할 때까지의 공백을 메우기 위해 위치 예측을 수행하여 모든 클라이언트에게 브로드캐스팅을 하게 된다. 긴 패킷 전송 간격으로 인한 예측 오차를 줄이기 위하여 전송 간격 최대한계치와 이중 패킷전송 등의 추가적 작업을 수행한다. 결과의 효율성을 보이기 위해 테스트 게임 환경을 구축하여 기존의 고정된 패킷전송 주기 시스템과의 비교분석을 수행하였다.

▸ **주제어**: 클라이언트-서버 시스템, 서버측 시뮬레이션, 패킷전송, 네트워크 게임, 동기화

• First Author: Seungmuk Oh, Corresponding Author: Yoonsik Shim
*Seungmuk Oh (idea7654@naver.com), Dept. of Game Engineering, Pai Chai University
*Yoonsik Shim (ysshim@pcu.ac.kr), Dept. of Game Engineering, Pai Chai University
• Received: 2023. 01. 02, Revised: 2023. 02. 09, Accepted: 2023. 02. 10.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

최근 몇 년간 게임 시장에서 '배틀그라운드'를 중심으로 대규모 1인칭 온라인 슈팅 게임 장르가 세계 최대 규모의 전자 게임 소프트웨어 유통망인 'Steam'에서 동시 접속자 순위 상위권을 차지하며 주목받고 있다 [1]. 그러나 1인칭 온라인 슈팅 게임 장르의 선구자인 '카운터 스트라이크'의 사례에서와 같이 클라이언트-서버 구조를 통한 1인칭 온라인 슈팅 게임 장르는 잦은 상태 변화로 인해 많은 양의 패킷이 송수신되며 서버의 네트워크 부하가 늘어나게 됨을 알 수 있다 [2]. 네트워크의 부하가 늘어난다는 것은 곧 서버가 필요로 하는 네트워크 대역폭이 늘어난다는 것을 의미하며 이는 게임 운영의 비용 증가로 이어진다. 이에 따라, 네트워크의 부하를 줄이면서도 효율적인 동기화를 위해 FSM (Finite State Machine)과 이벤트 잠금 (Event holding) 기법을 결합한 동기화 기법 [3], 임계각 설정을 통한 효율적인 캐릭터 방향 갱신 기법 등이 연구되었다 [4]. 따라서, 본 논문에서는 이러한 문제점을 완화하기 위하여 대규모 1인칭 온라인 슈팅 게임에서 최소한의 오차로 최대한의 네트워크 부하 감소를 위한 적응적 패킷전송 주기 방법을 제시한다. 클라이언트가 매 패킷을 전송할 때마다 플레이어의 위치 및 움직임 변수들의 변화량을 측정하여 이를 기반으로 다음번 패킷이 전송되어야 할 시간 간격을 계산함으로써 고정된 전송 주기가 아닌, 상황에 따라 변화하는 패킷전송 간격을 구현하며, 간격이 넓은 패킷전송 상황에서의 예측 오차를 줄이기 위하여 전송 간격 최대 한계치와 이중 패킷전송 등의 추가적 방법을 함께 제시한다. 2장에서의 관련연구를 설명한 후, 3장에서는 본 논문에서 제안한 방법을 세부 수식과 더불어 기술한다. 이어 4장에서는 고정주기 패킷전송 방식과 본 논문에서 제안하는 가변주기 패킷전송 방식을 같은 환경에서 실험을 진행하여 효율성을 입증한다.

II. Relative Work

1. Networked Physics

실시간으로 진행되는 게임의 경우, 접속해서 함께 게임을 하는 유저들 모두에게 동일한 경험을 선사하는 것을 목표로 한다. 동일한 경험을 선사하기 위해서는 모든 클라이언트가 같은 Physics 환경을 공유하여 다른 화면에서 같은 결과를 내놓도록 해야 한다. 네트워크로 연결되어있는 서로 다른 환경에서 같은 Physics 환경을 갖게 하는 것을 Networked Physics라 한다. 대규모 1인칭 온라인 슈팅 게임 중 하나인 'Apex Legends'의 경우 게임 내에서 진행되는 대부분의 연산이 서버에서 이루어지기 때문에 서버에서 매 프레임에서의 상태 정보를 위한 연산을 진행하며, 이러한 프레임 연산의 마지막 단계에서 상태들의 스냅숏을 저장한 후 브로드캐스팅하는 방식[5]을 사용하고 있다. 서버에서 상태 예측을 진행할 때 사용하는 또 다른 방법으로는 동일한 초기조건 하에서 동일한 입력을 받을 때 서버의 상태 예측이 동일한 상태 결과를 내놓도록 하는 Deterministic Lockstep 기법이 있다 [6]. Deterministic Lockstep은 위치, 방향 등의 상태는 전송하지 않고 Table 1에서 볼 수 있듯이 플레이어의 Input만을 전송한다. 따라서 주로 마우스를 통해 입력하게 되는 1인칭 온라인 슈팅 게임의 특성상 시시각각 마우스의 회전속도가 급격하게 변하는 상황에서 Deterministic Lockstep 기법을 적용할 때는 마우스의 sampling rate를 낮추는 동시에 위치 예측 시 발생하는 오차의 리스크를 줄이기 위해서 한 세션 당 적은 인원의 플레이어 수로 제한하게 된다. 따라서 본 논문에서도 스냅숏 배포 기반으로 구현하는 과정에서의 오차를 최소화하면서도 서버에서 위치 예측을 진행하는 동안 네트워크 부하를 감소를 위한 방법을 제시한다.

Table 1. Input for deterministic lockstep

Data Type	Input
bool	Left
bool	Right
bool	Up
bool	Down
bool	Space
bool	Z

2. Dead Reckoning

온라인 게임에서의 서버와 클라이언트들은 플레이어들 간의 동기화 또는 상태 갱신을 위해 패킷들을 주고받는데, 이때 서버에 전송된 후 모든 클라이언트에 퍼지는 방식(클라이언트-서버 구조) 또는 다른 클라이언트들에게 직접 전송하는 방식(피어 투 피어 구조) 중 어떤 방식을 사용하더라도 Fig. 1과 같이 패킷전송에 걸리는 물리적인 시간이 발생하게 된다. 이 때문에 각 클라이언트별로 게임 상태의 불일치가 생기게 되는데, 이러한 불일치를 최소화하는 방법 중 하나가 데드레커닝이다. 데드레커닝은 서버가 클라이언트로부터 마지막으로 전송받은 위치와 속도 및 가속도 등의 데이터를 이용하여 해당 플레이어가 현재 어디에 있는지를 예측하는 방법이다 [7]. Table 2는 세 가지의 서로 다른 데드레커닝 알고리즘의 예를 보여주고 있다. 첫

번째 알고리즘은 플레이어의 다음번 위치 정보를 받을 때까지 이전 위치를 유지하는 알고리즘이다. 두 번째 알고리즘은 마지막으로 패킷을 받은 시간 t_0에서의 위치에서 플레이어의 속도 v_0에 기반하여 다음 시간의 위치(시간 t_1에서의 위치)를 추정한다. 세 번째 알고리즘 또한 마지막으로 패킷을 받은 위치로부터 전방으로 추정하지만, 속도와 가속도(a_0)를 모두 사용한다. FPS 게임을 다루는 본 논문에서는 속도만 고려하는 두 번째 방법을 사용한다.

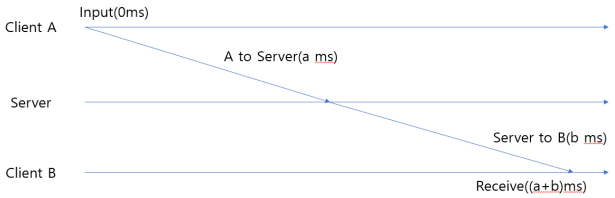

Fig. 1. Simulation mismatch by network latency

Table 2. Dead Reckoning Algorithms[8]

	Algorithm
1	$Pos_1 = Pos_0$
2	$Pos_1 = Pos_0 + v_0(t_1 - t_0)$
3	$Pos_1 = Pos_0 + v_0(t_1 - t_0) + 0.5a_0(t_1 - t_0)^2$

3. Client-Side Prediction

Client-Side Prediction (and Server Reconciliation)은 게임의 determinism이 보장되면 사용자의 입력 정보를 바탕으로 위치를 예측하는 방식이다. 본래 Client-Side Prediction은 클라이언트가 입력을 보낸 후 렌더링할 새로운 상태를 기다리는 대신에 클라이언트도 자신만의 위치 예측을 진행하여 입력 지연을 없애는 방식이다[9]. 이때, 플레이어의 위치 예측은 마지막으로 받은 입력 정보에 기반하여 레이턴시만큼의 위치를 더한 상태를 예측하게 된다. 본 논문에서도 Client-Side Prediction에 사용되는 예측 방법을 서버 측 예측에 사용하여 실험을 진행한다.

III. Propose Method

1. Client

1-1. Client-side Variables

Table 3은 클라이언트에서 패킷을 보낼 때 고려해야 할 환경변수이다. 위 변수들을 통해 클라이언트는 언제 자신이 패킷을 보낼지 계산하여 매 프레임 자신이 패킷을 보내야 하는지를 판단한다. 다음 패킷을 보낼 시간 간격은 패킷을 보낸 후 최대 허용 interval인 1초로 초기화되며 보낸 이후에 매 프레임 이전 패킷을 보낸 상태와 비교하여 거리가 멀어질수록, 차이각이 클수록, 또 방향차가 클수록 주기가 짧아지게 된다. 이는 즉 이전 상태에 비해 상태 변화가 클수록 패킷을 빠르게 보내어 자신의 상태를 갱신하여 서버 측 위치 예측에 따른 오차를 최소화하기 위함이다.

Table 3. Attributes for Movement Variability

State	Description
d	Distance between the current position and the previously transmitted positions
α	The angle between the current forward vector and the vector at the previous transmission time
U	Movement vector difference between the current time and the previous transmission time

1-2. Packet Transmission Interval

패킷을 보낼 주기를 측정하는 방법은 다음과 같다. 클라이언트에서 ms단위로 측정된 현재 시간(t)과 이전 패킷을 전송한 시간(T_b)이 주어졌을 때 위치 차와 전방 벡터 차는 다음과 같다.

$$d = |\mathbf{p}(t) - \mathbf{p}(T_b)| \quad (1)$$

$$\alpha = (1 - \mathbf{r}(t) \cdot \mathbf{r}(T_b))/2 + 1 \quad (2)$$

$\mathbf{p}$는 캐릭터의 위치이며 $\mathbf{r}$는 캐릭터의 전방 방향 노멀벡터이다. 캐릭터의 이동 방향의 차이 U의 계산은 다음 식에 따라 시간 t에서 측정된 게임 tick인 n과 바로 이전 tick인 $n-1$을 이용하여 정규화된 이동 방향 벡터들을 얻고 그 내적을 구함으로써 일치도를 계산한다.

```
[Algorithm Pseudocode]

WHILE
    IF (currentTime - prevSendTime > 1.5)
    THEN
        isPacketLoss = true
    ENDIF

    IF (isPacketLoss == false)
    THEN
        #Predict Player Position
    ENDIF

    #Update Game States
```

Fig. 2. Prediction and State Update Loop for Server

$$\mathbf{u}(T_b) = N[\mathbf{p}(n_b) - \mathbf{p}(n_b - 1)] \quad (3)$$

$$\mathbf{u}(t) = N[\mathbf{p}(n) - \mathbf{p}(n-1)] \quad (4)$$

$$U = (1 - \mathbf{u}(T_b) \cdot \mathbf{u}(t))/2 + 1 \quad (5)$$

함수 $N[\ ]$은 벡터 정규화를 나타내며, 이동 방향 벡터는 플레이어가 이동하고 있는 속도에 대한 월드 좌표상에서의 단위벡터를 나타낸다. 캐릭터의 전방 방향 차이 α와 이동 방향 차이 U의 값은 두 방향 벡터가 일치할수록 1로, 멀수록 2를 나타내도록 조절되었다. 식 (1)~(5) 에서 구한 위치, 전방 방향, 이동 방향의 차이들을 이용해 현재 시간으로부터 다음번 패킷이 전송되기까지 소요되어야 할 시간을 아래와 같이 계산한다.

$$T = 2 / (1 + \exp(k \times d \times \alpha \times U)) \quad (6)$$

T는 이전 패킷을 보낸 시간(T_b)부터 다음 패킷을 보내기까지 걸려야 할 시간이며, k는 기울기 상수로, 큰 값을 넣을수록 플레이어의 상태 변화에 민감하게 작동하여 패킷을 보내는 주기를 짧게 할 수 있다. k는 게임의 특성에 따라 적절한 값을 상수로 정하게 되며, k의 값을 작게 하면 오차가 커질 확률이 증가하기 때문에 서버의 tick 값이 큰 경우 또는 서버 시뮬레이션에 있어 오차를 많이 허용할 수 있는 경우 작은 값을 사용할 수 있다. 그러나, 서버의 tick이 낮아 서버 시뮬레이션의 위치 갱신에 더 많은 시간이 걸리거나 오차를 최대한 적게 허용해야 할 경우에는 k의 값을 크게 설정하여야 한다. 캐릭터의 위치가 변하지 않는 상황에서는 나머지 상태 차와 무관하게 최대 허용 interval인 1초 주기로 패킷을 보내게 되는데 이때 서버에서는 지속적으로 클라이언트로부터 받은 속도와 위치를 기반으로 다음 위치를 예측하고 있기 때문에 플레이어가 중간에 멈췄을 때는 즉각적으로 멈췄다는 정보를 알려주는 패킷을 보내어 서버에서의 추가적인 계산을 유예하도록 한다. 실제 패킷전송은 클라이언트의 프레임 tick과 실제 측정된 시간의 균일성을 고려하여 다음과 같은 식을 기준으로 구현된다.

$$P = t - T_b - T \quad (7)$$

위와 같이 클라이언트는 매 프레임 P를 측정하여 P가 0 이상일 때 패킷을 보낸 후 T_b의 값을 t로 재설정한다. 반대로 P가 0보다 작을 경우엔 패킷을 전송하지 않는다. 이렇게 평균 패킷 송신 주기가 클라이언트 측의 상태변화량에 반비례하도록 함으로써 Client-Side Prediction 기반 위치 예측의 오차를 최대한 줄일 수 있도록 하였다.

2. Player Position Prediction in Server

2-1. Client-Side Prediction

본 논문에서 제안하는 방법을 사용한 멀티플레이어 1인칭 온라인 슈팅 게임의 PvP 상황에서의 판정 등을 위하여 서버는 클라이언트로부터 패킷을 받아 해당하는 플레이어의 상태를 업데이트한 후, 다음 패킷을 받을 때까지 매 서버 측 프레임마다 클라이언트의 위치를 시뮬레이션하여야 한다. 실시간으로 예측하는 시뮬레이션 서버의 매 프레임 시간을 t_n이라고 하였을 때 Client-Side Prediction 알고리즘을 통하여 다음과 같이 클라이언트의 현재 위치를 예측할 수 있다.

$$\mathbf{Pos}_1 = \mathbf{Pos}_0 + \mathbf{u}_{kb} * v(t_1 - t_0) \quad (8)$$

$\mathbf{Pos}_1$은 현재의 위치벡터이며, $\mathbf{Pos}_0$은 마지막으로 받은 플레이어의 위치이다. $\mathbf{u}_{kb}$는 플레이어의 전방 방향을 기준으로 8방향으로 가능한 키보드 입력 정보로부터 얻어진 단위벡터로서 서버가 패킷을 받을 당시 클라이언트가 진행하려 의도한 방향을 나타낸다. 해당 정보를 바탕으로 서버는 이전에 받은 위치 $\mathbf{Pos}_0$와 마지막으로 받은 이동 속력 v를 사용하여 현재 캐릭터의 위치를 예측하게 된다. 해당 방식을 사용할 때 패킷이 몇 프레임 동안 도착하지 않은 플레이어의 위치 정보의 정확성이 문제가 될 수 있는데, 이를 위해 식 (6) 에서의 클라이언트 패킷전송 주기를 위한 기울기 상수 k를 조정하여 게임별로 설정할 수 있게 하였다. 상수 k가 커질수록 플레이어가 비선형적인 움직임을 취했을 때 패킷을 자주 보내게 되기 때문에 선형적인 움직임을 취했을 때는 보다 효율적으로, 비선형적인 움직임을 취했을 때는 일정 간격으로 패킷을 보내는 주기와 비슷해져 위치 예측의 오차를 최소화할 수 있다. 또한, 비선형적인 움직임에서 선형적인 움직임으로 전환하는 순간 급격히 길어진 T로 인한 서버 측 위치 예측의 간헐적인 큰 오차를 방지하기 위해 클라이언트는 매 패킷을 보낸 뒤 0.05초 후에 항상 다시 한번 패킷을 전송하여 이러한 경우의 예측에 있어 오차를 최소화하도록 하였다.

2-2. Packet Loss

1인칭 온라인 슈팅 게임은 보통 짧은 간격으로 많은 패킷을 전송하기 때문에 심각한 수준의 패킷 손실이 발생하지 않는 한, 일시적인 패킷 손실의 경우 크게 문제가 되지 않는다. 그러나 본 논문에서 제안하는 방법을 사용할 때,

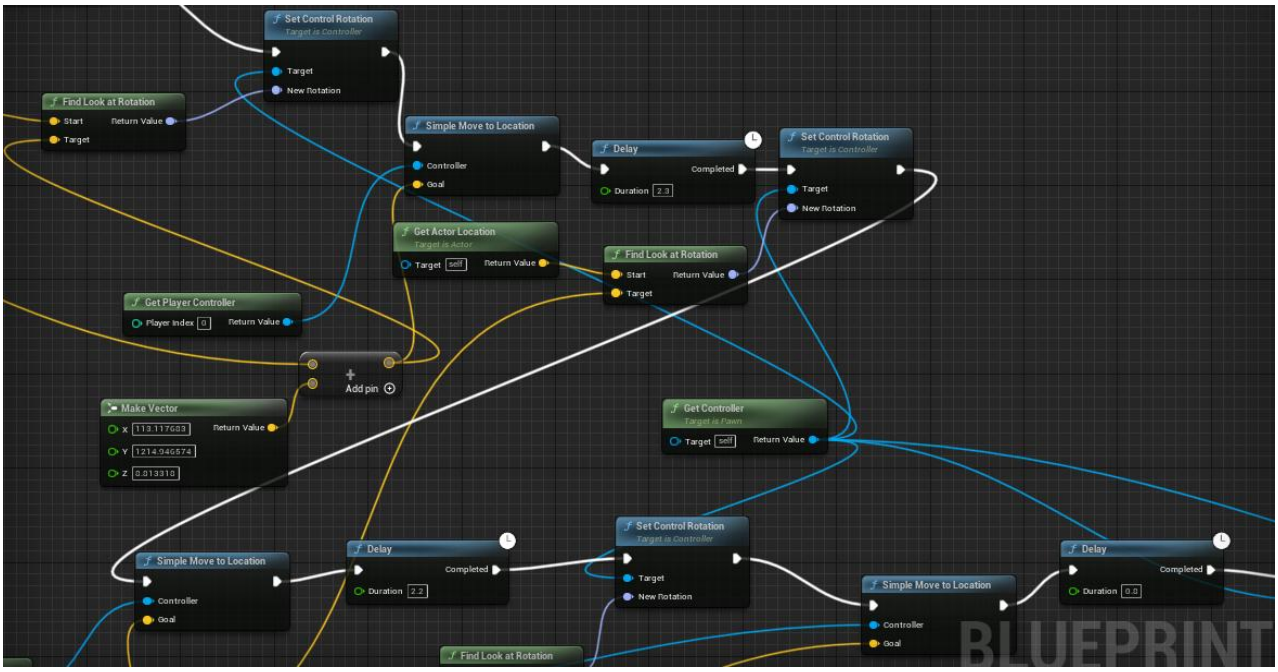

Fig. 3. Unreal Engine blueprint for player movement logic

클라이언트 측의 선형적인 움직임으로 인하여 길어진 패킷전송 주기가 지속되는 상황에서의 일시적인 패킷 손실은 서버의 예측 오차를 크게 할 가능성이 있다. 따라서 이러한 경우에 패킷 손실에 따른 예측 오류를 최소화하기 위하여 서버 측에게서는 임계시간(c=1.5초)을 설정하여 접속한 플레이어들이 마지막으로 보낸 패킷의 시간과 현재 시간의 차를 계산하여 임계시간을 넘은 플레이어에 대해서는 패킷 로스 발생을 나타내는 변수(Table 5: isPacketLoss)를 true로 변경하여 위치 예측을 중지함으로써 마지막으로 예측되었던 위치에 플레이어를 고정 시킨다. 이후 새로운 패킷이 도착했을 때 해당 위치로 텔레포트 시킨 후 패킷 로스 변수를 다시 false로 변경한다.

IV. Experiments

1. Test Environment

본 논문의 방법을 위한 실험환경은 서버-클라이언트 구조로 구성하였으며, 클라이언트 로직은 언리얼 엔진 5의 블루프린트를, 서버는 C++과 TCP를 사용하였다. 언리얼 엔진 5에서는 Fig. 3과 같은 로직을 통해 플레이어가 자동으로 정해진 경로를 이동하거나 회전하도록 하여 시도마다 항상 동일한 움직임이 가능하게 하였다. 서버 측 시뮬레이션 스레드는 20Hz로 작동하며, 본 논문에서 제안한 방식으로 진행한 실험의 경우에는 매 프레임 캐릭터의 위치 예측을 실행하였다 (Eq. (8)). 고정된 클라이언트 패킷 전송 주기를 가진 '카운터 스트라이크' 게임 연구[2]와 같은 조건의 실험에서는 서버 시뮬레이션 갱신 속도 이상의 패킷 전송량을 사용하기 때문에 단순히 매 프레임에서의 캐릭터의 마지막 위치를 사용하였다.

Table 4. Packet Structure

State	Description
position	Character position vector
moveSpeed	Moving speed (v)
Input	Player input state structure
transTime	Time of packet transmission

Table 5. Player States in Server

State	Description
position	Predicted position from the last received position: $\mathbf{Pos}_1$ in Eq (8)
prevPos	Previously received position: $\mathbf{Pos}_0$ in Eq (8)
moveSpeed	Previously received moving speed (v)
prevSendTime	Time of the last packet reception
isPacketLoss	Packet loss flag

2. Experiment Procedure

모든 실험에서는 동일한 클라이언트 이동 경로 및 로직을 사용하였다. 한쪽에서는 일정 주기로 자신의 위치 정보를 전송하여 일정 주기로 자신의 위치를 서버에 알리고,

서버는 위치 정보를 받아 해당 플레이어 객체의 위치 정보를 업데이트한다. 이때, 서버에서는 통신 스레드와 별도로 상태 시뮬레이션 서버가 20Hz로 작동하여 매 프레임 클라이언트의 위치를 기록한다. 다른 한쪽에서는 제안한 방식에 따라 서버에게 자신의 상태를 전송한다. 이때, 서버 측에게서는 플레이어의 상태를 사용하는 것이 아니라 Client-Side Prediction을 통한 위치 예측을 통해 예측한 위치를 사용한다. 이후, 두 플레이어의 위치 정보를 분석하여 최대 오차와 전송한 패킷량을 비교하여 본 논문에서 제안한 방법의 효율성을 입증하였다. 클라이언트에서 보내는 패킷의 내용은 Table 4와 같다. Table 5는 서버에서 고려하는 변수들을 담아놓은 플레이어 상태 클래스이며, 각 플레이어는 클래스의 인스턴스 형태로 존재하도록 하여 실험을 진행하였다. 실험은 총 2개로 나누어 진행되었고, 같은 환경에서 기울기 상수 k값을 조정하면서 각 실험 결과를 기록하였다.

3. Experiment Result

Fig. 4는 초당 20회씩 고정주기 패킷을 보내어 서버 시뮬레이션 시 각 프레임의 플레이어 위치를 나타낸 것이다. 초당 20회라는 수치는 잘 알려진 1인칭 온라인 슈팅 게임 중 하나인 '카운터 스트라이크'를 기준으로 1초에 약 22개

Table 6. Total packet transmission

Experiments	Total Packet
fig. 4	285~295
fig. 5	225~235
fig. 6	170~180

Table 7. Position error rate

Experiments	Max Error	Average Error
fig. 4 & fig. 5	8%	1%
fig. 4 & fig. 6	10%	2%

의 패킷을 전송하는 경우와의 효율성을 비교하기 위해 설정한 주기이다. Fig. 5와 Fig. 6은 본 논문에서 제안한 방식에 따른 서버 시뮬레이션 시 각 프레임에서 예상된 플레이어의 위치를 나타낸 것이다.

Fig. 5는 기울기 상수 k를 10으로 설정하였고, Fig. 6은 기울기 상수 k를 6으로 설정하였다. 이때, Fig. 4, 5, 6은 모두 서버의 프레임, 클라이언트의 움직임이 같은 환경에서 실험되었으며 Fig. 5에서의 패킷 전송량은 Table 6과 같이 Fig. 4에 비하여 약 20%가 감소하고, Fig. 6에서의 패킷 전송량은 Fig. 4에 비하여 약 40%가 감소하여 플레이어가 선형적인 움직임을 보일 때 특히 큰 효율을 보임을 확인할 수 있었다. 반면 오차의 경우에는 Table 7과 같은 결과를 보였다. 오차는 같은 프레임에 해당하는 각 Fig. 4의 좌표와 Fig. 5의 좌표, 그리고 Fig 4와 Fig. 6의 좌표의 유클리드 거리를 계산하여, 고정주기를 사용하는 Fig 4와 비교하였을 때 이동속도에 비해 오차가 얼마나 큰지를 %로 나타낸 것이다.

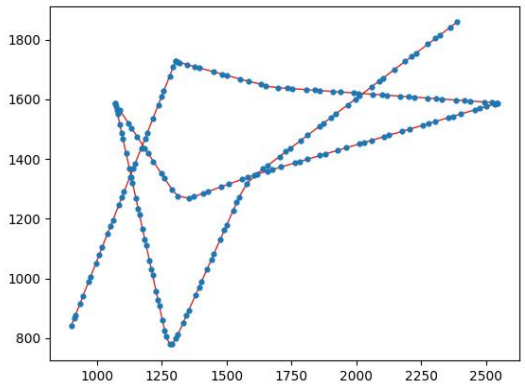

Fig. 4. Position traces from fixed-rate packet transmission

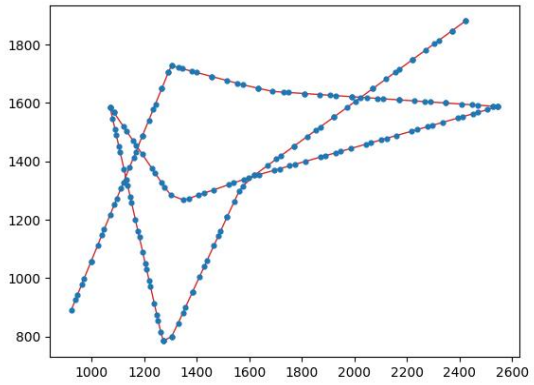

Fig. 5. Traces from adaptive transmission for k=10

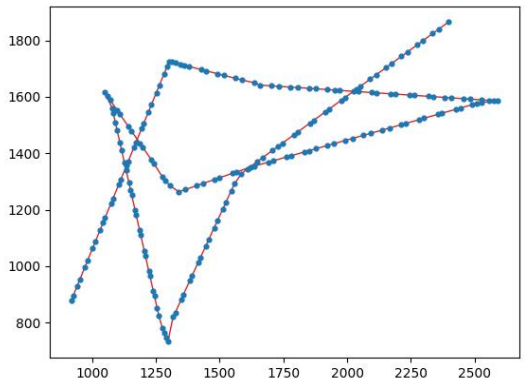

Fig. 6. Traces from adaptive transmission for k=6

V. Conclusions

대규모 1인칭 온라인 슈팅 게임은 정확한 시뮬레이션이 중요하며, 네트워크 부하를 줄임과 동시에 플레이어에게 쾌적한 환경을 제공하는 목표를 갖고 있다. 본 논문에서는 대규모 1인칭 온라인 슈팅 게임 장르에서 최소한의 오차로 최대한의 네트워크 부하 감소를 위한 방법을 제시하였다. 알고리즘을 제시하는 과정에서 클라이언트와 서버에서 각각 고려되는 변수들을 살펴보았으며 클라이언트에서는 이전 상태와의 거리 차이, 각도 차이, 방향 차이 등을 이용하여 효율적으로 패킷을 전송하는 방법을 제안하였다. 서버 쪽에서는 클라이언트로부터 전송받은 캐릭터들의 상태를 바탕으로 각 플레이어의 위치를 안정적으로 예측하기 위하여 Client-Side Prediction 방법과 함께 움직임 상태의 질적 변화 및 패킷 손실 경우를 위한 부수적인 루틴을 추가로 사용하였다. 이어 본 논문에서 제안한 방식으로 실험 환경을 제작하여 실험을 진행한 결과, 플레이어가 선형적인 움직임을 보일 때 패킷 전송량에 있어 큰 효율을 보임을 알 수 있었다. 오차의 임계치를 조절하기 위해서는 클라이언트에서 제안한 공식의 기울기 상수 k를 조절하여 설정할 수 있었고, 결과적으로 대규모 1인칭 슈팅 게임 장르 게임에서 고정된 주기로 패킷을 보내 시뮬레이션을 진행하는 방식보다 효율적임을 입증하였다.

VI. Future Work

본 논문에서는 서버 시뮬레이션을 위한 효율적인 패킷 전송 방법을 제안하고 클라이언트와 서버에서 여러 변수를 통해 플레이어의 위치를 서버에서 예측하여 시뮬레이션에 사용함으로써 서버 측에서 수신받는 패킷의 양을 줄일 수 있었다. 그러나 서버에서 게임을 시뮬레이션할 때는 레이턴시로 인하여 각 플레이어가 경험하는 환경에 차이가 발생할 수 있는데 이를 위하여 Entity Interpolation [10], Lag Compensation[11] 등의 기술들을 적용하여 결과의 개선을 꾀할 수 있다. 또한 서버에서 매 프레임 시뮬레이션을 진행한 후 스냅숏을 배포할 때 송신의 대역폭을 줄이기 위하여 우선순위를 사용하거나 구역 분할 등을 통해 송신 대역폭을 줄이는 기술들도 적용할 예정이다[12]. 마지막으로 게임마다 허용되는 위치 예측의 최대 오차가 다른 점들을 고려하는 동시에 본 논문에서 제안한 방식과 함께 앞서 나열한 기술들을 결합하여 송수신 대역폭을 낮추고 플레이어에게 쾌적한 환경을 제공할 수 있도록 실제 게임에 적용하고 효율성을 입증하는 것이 필요하다.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2020R1G1A1101137).

REFERENCES

[1] Adam Bankhurst, Steam Reveals Its Best-Selling and Most Played Games of 2021, https://www.ign.com/articles/steam-reveals-its-best-selling-and-most-played-games-of-2021

[2] Feng. W. et al. “Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server,” ACM SIGCOMM Computer Communication Review, Vol. 32, Issue 3, pp. 18, July. 2002. DOI: 10.1145/571697.571707

[3] Hye-young Kim, “A Synchronized Scheme Applying on Hybrid in On-Line Game,” The Journal of The Institute of Internet, Broadcasting and Communication, Vol. 11, No. 2, pp. 7-12, November. 2011. DOI: 10.7236/JIWIT.2011.11.2.007

[4] JongMin-Lim, Dongwoo-Lee, Youngsik-Kim, “An Efficient Method to Update Character Moving Directions for Massively Multi-player Online FPS Games,” Journal of Korea Game Society, Vol. 14, No. 5, pp. 35-42, Oct. 2014. DOI: 10.7583/JKGS.2014.14.5.35

[5] Samy Duc, WHAT MAKES APEX TICK: A DEVELOPER DEEP DIVE INTO SERVERS AND NETCODE, https://www.ea.com/games/apex-legends/news/servers-netcode-developer-deep-dive?setLocale=en-us

[6] Zuohao Yan, Zhihan Lv, “The Influence of Immersive Virtual Reality Systems on Online Social Application,” Applied Sciences, Vol. 10. No. 15, pp. 5058, July 2020. DOI: 10.3390/app10155058

[7] Wei Shi, Jean-Pierre Corriveau, Jacob Agar, “Dead Reckoning Using Play Patterns in a Simple 2D Multiplayer Online Game,” International Journal of Computer Games Technonlogy, Vol. 2014, January. 2014. Article No. 5, pp 5, DOI. 10.1155/2014/138596

[8] Youfu Chen, Elvis S. Liu, “Comparing Dead Reckoning Algorithms for Distributed Car Simulations,” Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. pp. 105-111, May 2018. DOI: 10.1145/3200921.3200939

[9] Takato Motoo, Jiei Kawasaki, Takuya Fujihashi, Shunsuke Saruwatari, Takashi Watanabe, “Client-Side Network Delay Compensation for Online Shooting Games,” IEEE Access. Vol. 9, pp. 125678-125690, August 2021. DOI: 10.1109/ACCESS.2021.3111180

[10] Luke Larson, “Astrophobia: A 3D Multiplayer Space Combat Game with Linear Entity Interpolation,” Califonia Polytechnic

State University, Bachelor of Science in Computer Science. December 2013.

[11] Zhi Li, Hugh Melvin, Rasa Bruzgiene, Peter Pocta, Lea Skorin-Kapov, and Andrej Zgank, “Lag Compension for First-Person Shooter Games in Cloud Gaming,” Autonomous Control for a Reliable Internet of Service, Lecture Notes in Computer Science, Vol. 10768, pp. 104-127. January 2018. DOI: 10.1007/978-3-319-90415-3_5

[12] Glenn Fiedler, State Synchronization, https://gafferongames.com /post/state_synchronization/

Authors

Seungmuk Oh received the Bachelor of Science in Engineering degree in Game Engineering from Paichai University, Korea, in 2023. He is interested in network game programming, parallel computing and computer graphics application.

Yoonsik Shim received the B.S. in Mechanical Engineering and M.S. in Computer Science from Korea University, Korea. He received the Ph.D. degree in Informatics from University of Sussex, UK, in 2013. Dr. Shim is currently an Assistant Professor in the Department of Game Engineering, Pai Chai University. His research interests are bio-inspired adaptive robotics, computational neuroscience, chaotic neurodynamics, self-organization, and evolutionary robotics.