Journal of Korea Game Society 2015 Dec; 15(6): 97-106
http://dx.doi.org/10.7583/JKGS.2015.15.6.97

게임 프로그래밍

다중 접속 온라인 게임을 위한 유니티 엔진의 네트워크 스레드 패킷 폐기 기법

유종근, 김영식
한국산업기술대학교 게임공학과
neoracid@naver.com, kys@kpu.ac.kr

Packet Discard Policy of Network Thread in an Unity Engine for Multi-player Online Games

Jong-Kun Yoo, Youngsik Kim
Dept. of Game and Multimedia Engineering, Korea Polytechnic University

요 약

유니티 엔진 기반 다중 접속 온라인 게임에서는 통신 패킷 처리를 담당하는 네트워크 스레드와 게임 로직을 담당하는 메인 스레드를 분리해야 한다. 네트워크 스레드 간에 통신 패킷은 렌더링 속도 향상을 위하여 중복되는 패킷을 폐기할 필요가 있다. 본 논문에서는 유니티 엔진 기반 다중 접속 온라인 게임을 위한 네트워크 스레드 간 통신 패킷 폐기 기법을 제안한다. 제안하는 기법은 Partial Packet Discard 기법과 Periodic Packet Discard 기법을 혼합한 기법으로 네트워크 패킷을 큐로 관리하고 중복되는 패킷을 주기적으로 폐기하여 렌더링 속도를 향상시킨다. 제안하는 기법은 유니티 엔진 기반 다중 접속 온라인 게임의 다양한 패킷 발생 시뮬레이션을 통하여 렌더링 속도를 분석하고 효율성을 증명하였다.

ABSTRACT

In an Unity engine for multi-player online games, the main thread processing game logic must be separated from the network thread that is responsible for network packet communication. Packet communication between the network threads needs to drop packets that overlap in order to improve the rendering speed. In this paper, the packet discard policy of network thread is proposed for an Unity engine for multi-player online games. The proposed method is the hybrid method of both Partial Packet Discard and Periodic Packet Discard methods to improve the rendering speed by periodically discarding overlapped network packets managed by the queue. The rendering speed of the proposed method is analyzed and its effectiveness is verified by various packet generating simulations of the Unity engine for multi-player online games.

Keywords : Multi-player Online Game (다중 접속 온라인 게임), Unity Engine (유니티 엔진), Network Packet Discard (네트워크 패킷 폐기)

Received: Sep, 09, 2015 Revised: Oct, 07, 2015
Accepted: Nov, 16, 2015
Corresponding Author: Youngsik Kim (Korea Polytechnic University)
E-mail: kys@kpu.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

한국에서의 온라인 게임 산업의 경우 2013년 기준 온라인 게임시장의 규모는 5조 4천 532억원으로 한국 게임 시장 중에 56.1%를 차지하였다[1]. 과거의 게임 개발은 대부분 게임에 필요한 모든 시스템을 직접 개발을 하지만, 최근에는 게임 엔진의 발전으로 필요한 특정 게임 시스템 부분들만 개발을 하면 되기 때문에 개발에 필요한 시간이 절반 이하로 줄어들었다. 그 중에 유니티 엔진은 비용 대비 우수한 그래픽 요소와 시스템 개발 환경을 제공하여 게임 개발 툴로 많은 개발자들이 사용하고 있으며 세계적으로 전체 개발자의 45%가 유니티 엔진을 사용하고 있다[2,3,4].

최근 네트워크 온라인 게임과 관련된 연구가 많이 진행되었다[5,6,7]. 논문[5]에서는 논타켓팅 AOS (Aeon of Strife : 대전 액션과 공성전이 결합된 게임 장르) 온라인 게임을 설계하고 게임 서버는 IOCP 모델의 멀티 스레드로 제작하여 많은 클라이언트들을 수용할 수 있도록 하였다. 논문[6]에서는 대규모 온라인 FPS (First Person Shooter) 게임에서 효율적인 캐릭터 방향 갱신 기법을 제안하였다. 논문[7]에서는 다양한 장르의 네트워크 게임 트래픽을 분석하고 모델링하였다.

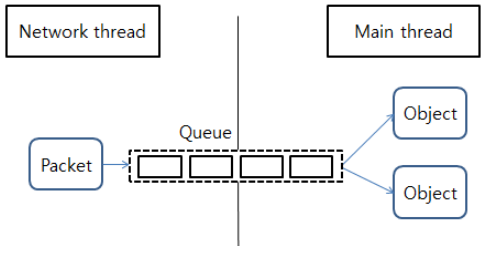
유니티 엔진 기반 다중 접속 온라인 게임에서는 통신 패킷 처리를 담당하는 네트워크 스레드와 게임 로직을 담당하는 메인 스레드를 분리해야하기 때문에 유니티 엔진 기반 게임 개발에 있어서 네트워크에 연관된 기법 및 알고리즘은 상당히 중요하다. 네트워크 스레드 간에 통신 패킷은 성능 향상을 위하여 중복되는 패킷을 폐기하는 많은 연구가 진행되었다[8,9,10].

본 논문에서는 유니티 엔진 기반 다중 접속 온라인 게임을 위한 네트워크 스레드 간 통신 패킷 폐기 기법을 제안한다. 제안하는 기법은 Partial Packet Discard 기법[8]과 Periodic Packet Discard 기법[9]을 혼합한 기법으로 네트워크 패킷을 큐로 관리하고 중복되는 패킷을 주기적으로 폐기하여 혼잡한 네트워크 상황에서의 빠른 패킷처리와 적은 지연시간을 가지며 렌더링 속도를 향상시킨다. 제안하는 기법은 자체 제작한 유니티 엔진 기반 다중 접속 온라인 게임의 다양한 패킷 발생 시뮬레이션을 통하여 렌더링 속도를 분석하고 효율성을 증명하였다.

2. 관련연구

2.1 유니티 엔진 기반 다중 접속 게임

유니티 엔진의 경우 메인 로직 스레드를 제외한 스레드에서 유니티 엔진 관련 함수를 쓰는 것을 막아 놓았다. 그래서 네트워크로 받은 패킷으로 객체를 관리해야하는 경우 네트워크 스레드와 메인 스레드 간의 공통의 자원관리 체계를 만들어 사용해야한다. 본 논문에서는 [Fig. 1]과 같이 일반적인 자료구조인 큐(queue)를 사용한다.



[Fig. 1] Thread Structure

네트워크 스레드는 패킷을 받으면 패킷 해석 후에 사전에 약속된 구조체로 변환 후 큐에 삽입한다. 이에 반응하여 메인 스레드는 매 프레임마다 실행되는 [Fig. 2]의 Update함수에서 큐를 이용하여 객체를 관리 한다. 그러나 네트워크가 혼잡해지는 상황에서는 [Fig. 2]의 Update 함수에서의 패킷 처리 속도보다 더 빠르게 패킷이 도착하기 때문에 큐에 패킷이 쌓이게 되고 이 과정에서 버퍼에 안에 패킷들은 처리가 지연된다. 흔히 온라인 게임 내에서 혼잡한 곳(대규모 사냥, 경매장, 레이드)의 경우 발생하게 된다.

Update Function Algorithm

```
queue_size = number_of_packets(queue)

for i from 1 to queue_size
        execute_packet(queue[i])
        destroy_packet(queue[i])
endfor
```

[Fig. 2] Update Function Algorithm

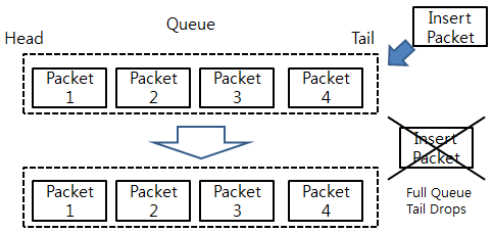
이러한 경우 게임의 렌더링 속도는 낮아지게 되며 최악의 경우 게임 접속이 끊기는 경우도 발생하게 된다. 그러므로 큐의 최대 크기를 제한하고 패킷 폐기 정책이나 큐 관리 기법 등을 활용하여 패킷을 폐기함으로써 게임 내 프레임을 적정선 유지시킬 수 있다. 그러나 기존의 패킷 폐기 기법들은 큐의 오버플로우를 막는 Drop head[10]방식으로 서비스가 제공되기 때문에 네트워크 요소에서 일정 수준으로 큐잉 지연시간을 보장하기가 어렵고 게임 내 패킷의 신뢰성을 보장하기 어렵다.

2.2 패킷 폐기 정책

2.2.1 Drop Tail

Drop Tail는 가장 간단한 구조이며, 선택적으로 패킷을 폐기하지 않고 새로운 패킷이 입력되었을 때 버퍼에 여유 공간이 없을 경우 새로운 패킷을 폐기하는 정책이다.

이 경우 문제점은 높은 우선순위인 패킷 즉 중요도가 높은 트래픽도 버퍼가 꽉 차 있다면 Drop을 시켜 손실될 수 있다는 것이다. 또한 패킷의 폐기율이 높지 않아 큐의 최대 크기는 제한할 수 있으나 성능은 높지 않다[8].

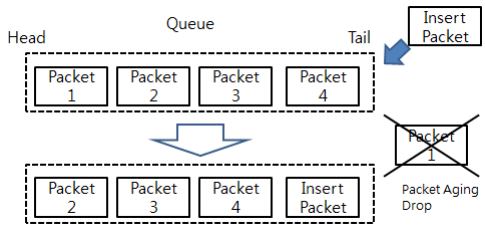


[Fig. 3] Drop Tail

2.2.2 Drop Head

Drop Head[10]의 경우 Drop Tail와 비슷하나 패킷의 폐기의 위치에 차이가 있다. Drop Tail의 경우 버퍼가 가득 찼을 경우 새로운 패킷을 폐기한다면 Drop Head의 경우 버퍼의 가장 앞에 패킷을 폐기하고 새로운 패킷을 버퍼의 가장 끝에 삽입한다.

Drop Tail와 높은 우선순위의 패킷이 Drop될 가능성을 가지고 있으나 실시간성이 중요한 게임에서는 새로운 패킷이 예전 패킷보다 우선순위가 높기 때문에 Drop Head를 많이 이용하며 가장 오래된 패킷을 폐기함으로써 Drop Tail보다는 게임 내 패킷의 신뢰성을 개선할 수 있다.



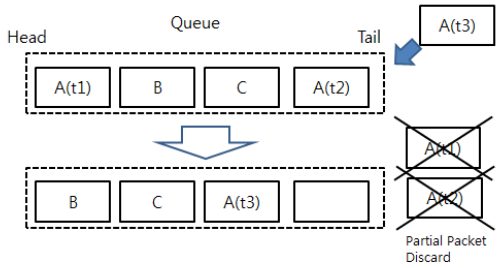
[Fig. 4] Drop Head[10]

2.2.3 Partial Packet Discard

온라인 게임의 경우 버퍼 안에 패킷들은 동일한 객체의 정보가 들어올 확률이 상당히 높다. 예를 들어 객체 A의 이동정보에 관한 패킷을 A(t)이라고 하였을 때 버퍼에는 A(t1),A(t2),A(t3) ..... A(tn)가 모두 들어 있을 것이다. 일반적인 게임플레이의 경우 객체 A에 관한 패킷들이 1 프레임 안에 처리되는 일은 거의 없으나 혼잡상태의 경우 버퍼의 패킷이 쌓여있기 때문에 동시에 처리가 필요할 때가 있다. 그런 경우 객체 A의 마지막 패킷 A(tn)을 제외 하고는 의미 없는 패킷이 되어 버리기 때문에 패킷 처리 기법인 Partial Packet Discard를 이용하여 패킷을 폐기한다.

Partial Packet Discard [8]은 Drop Tail를 개선한 형태로 셀을 폐기 할 때마다 동일한 패킷에 속하는 셀 중 마지막 셀을 제외한 모든 셀을 폐기한다. Partial Packet Discard에서 폐기된 패킷은

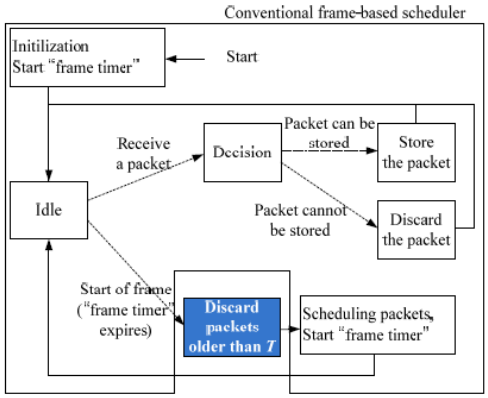
더 이상 전송되지 않으므로, 네트워크 사용 효율이 개선된다[8].



[Fig. 5] Partial Packet Discard[8]

2.2.4. Periodic Packet Discard

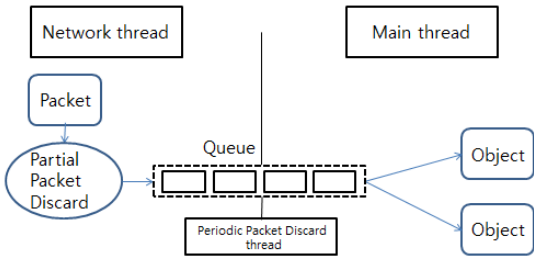
논문 [9]의 에서는 큐잉 지연시간을 줄이기 위한 주기적 패킷 폐기 기법(Periodic Packet Discard)을 제안하였다. 이 기법은 프레임 기반 스케줄러에서 스케줄링이 수행될 때 대기 시간 기반 패킷 폐기를 추가적으로 수행한다.



[Fig. 6] Behavior and transition condition of frame-based scheduler when periodic packet discard algorithm is performed after scheduling[9]

Periodic Packet Discard의 경우 스케줄링과 패킷 폐기의 순서에 따라 두 가지로 나눌 수 있다. [Fig 6]의 경우는 스케줄링 전에 패킷 폐기가 실행되는 경우이다. 주기적 패킷 폐기 알고리즘의 경우는 매 프레임의 시작마다 저장된 패킷들에 대해 스케줄링을 수행하고 나서 임계값(시간)을 초과한 시간동안 저장되어있던 패킷을 폐기한다. 다만 주기적 패킷 폐기 알고리즘 경우 패킷의 폐기를 단순히 버퍼에서 머문 시간만으로 판별하고, 임의로 패킷을 폐기하게 된다. 따라서 모든 사용자에 대해서는 성능 저하가 발생하게 된다[9].

3. 제안하는 네트워크 패킷 폐기 기법



[Fig. 7] Proposed Packet Discard Structure

유니티 엔진 기반 다중 접속 온라인 게임을 위한 스레드 간 패킷 폐기 기법을 제안한다. 제안하는 기법은 Partial Packet Discard[8]과 Periodic Packet Discard [9] 방식을 혼합한 방식이다.

제안하는 패킷 폐기 기법은 [Fig. 7]의 구조와 [Fig. 8]의 알고리즘과 같이 패킷이 네트워크 스레드로부터 입력이 들어왔을 때 Partial Packet Discard [8] 기법에 의해서 새로운 패킷과 동일 종류 패킷을 큐에서 폐기하고 새로운 패킷을 추가한다. 또한 여기서 기존에 생성되어있는 Periodic Packet Discard 스레드에 의해서 주기적으로 큐를 검사하여 지정된 시간이 지난 패킷을 폐기한다.

이 구조는 지속적으로 중복 정보를 가진 패킷을 폐기하고 또한 일정 시간이 지나 신뢰성을 잃은 패킷을 폐기하여 메인 스레드 부담을 줄여 그 결과 게임 내의 렌더링 속도를 향상 시킨다.

Proposed Packet Discard Algorithm

```
void Partial_Packet_Discard
{
  new_packet = network_read_packet()
  queue_size = number_of_packets(queue)

  for i from 1 to queue_size
    if queue[i].type == new_packet.type
          destroy_packet(queue[i])
    endif
  endfor
  add_packet(new_packet)
 }

void Periodic_Packet_Discard_Thread
{
 while(true)
    queue_size = number_of_packets(queue)

    for i from 1 to queue_size
      if queue[i].lifetime > current time
          destroy_packet(queue[i])
      endif
    endfor
 endwhile
}
```

[Fig. 8] Proposed Packet Discard Algorithm

4. 시뮬레이션 검증

본 논문은 다중 접속 온라인 게임에서 클라이언트가 네트워크 통신 패킷을 처리할 때 클라이언트로 입력되는 통신 패킷 폐기기법을 제안한다. 클라이언트 관점에서 고려한 네트워크 특성은 입력되는 패킷 생성률(packet per second), 패킷 처리 시간, 패킷의 종류 등의 네트워크 파라미터들을 가변화하여 렌더링 속도를 계산할 수 있도록 시뮬레이션 하였다. 따라서 네트워크 토폴리지는 1:N 의 온라인 환경을 가정한다. 제안하는 기법에 대한 성능 분석을 위해 [Fig. 8]에서 네트워크 패킷을 수신하는 네트워크 스레드를 일정시간마다 패킷을 생성하는 가상의 네트워크 스레드로 대체한다. 또한 패킷을 생성하는 시간을 조절하여 패킷량을 제어하고 그에 따른 가상의 네트워크 환경을 조성한다. 4.1절에서는 큐브와 같은 기본적인 프리미티브를 렌더링하는 기본적인 실험이다. 4.2절에서는 자체 제작한 유니티 엔진 기반 다중 접속 온라인 게임 환경에서 실험한 결과이다.

실험에 사용된 컴퓨터의 성능은 프로세서 : Inter(R) Core(TM) i5-2430M CPU @ 2.40GHz 2.40GHz, 메모리 : 4.00GB, 32비트 운영체제, 그래픽 카드 : AMD Radeon HD 7400M 이다.

4.1. 가상의 네트워크 환경 시뮬레이션

제안하는 기법에 대한 성능 분석을 위해 게임엔진 유니티를 활용하여 [Fig. 8]과 같이 구성하고 네트워크 스레드를 대체하기 위해 일정한 시간 마다 임의의 4종류의 패킷을 생성하는 스레드를 생성하여 가상의 네트워크 시뮬레이션 환경을 설정하였다.

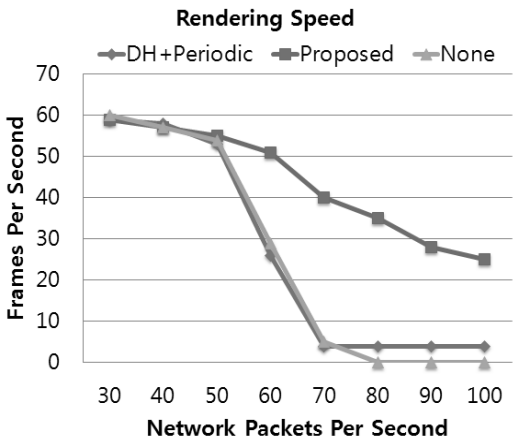
[Fig. 9]는 패킷처리시간이 15ms일 때 초당 패킷 생성률(packets per second)이 증가할 때 게임의 렌더링 속도 FPS(Frames Per Second)를 나타낸다. “none”은 아무 기법도 사용하지 않은 경우이며, “DH+Periodic”는 큐의 크기가 일정 크기를 넘어가면 Drop Head[10] 기법을 사용하였고 Periodic Packet Discard[9] 기법을 이용하여 생성시간이 일정이상 지난 패킷들을 폐기한다. “Proposed”는 제안하는 기법으로 Partial Packet Discard [8] 와 Periodic Packet Discard[9] 기법을 혼합 사용한 경우로써, 같은 종류의 패킷끼리는 폐기하고, 주기적으로 생성시간이 일정이상 지난 패킷들을 폐기한다. 또한 패킷의 폐기는 패킷 생성과 동시에 이루어지며 현재 큐의 크기와 큐의 구성을 관찰할 수 있고 실험변수로는 패킷의 생성시간과 패킷처리시간을 조정할 수 있다.

모든 경우에서 패킷의 생성률(packets per second)이 50 일 때 게임 렌더링 속도가 급격하게 감소하였고 “none”의 경우 패킷 생성률이 80 일 때 큐의 크기가 너무 커져서 렌더링 속도가 0

FPS로 측정이 불가능하였다.

패킷 생성률이 70 이상에서는 “DH+Periodic”의 경우 큐의 크기가 오버플로우 되지 않는 최대 크기가 되어서 렌더링 속도가 4 FPS로 매우 적게 유지된다. 그러나 “Proposed” 기법은 패킷 생성률이 70 이상의 경우에도 큐의 크기는 적정 크기를 유지하여 렌더링 속도가 다른 기법에 비하여 높다.

결론적으로 패킷 생성률이 70이하의 가상 시뮬레이션 환경에서 제안하는 기법이 “DH+Periodic”과 “none”에 비하여 각각 평균 2.0배와 1.6배 높은 렌더링 속도를 제공한다.

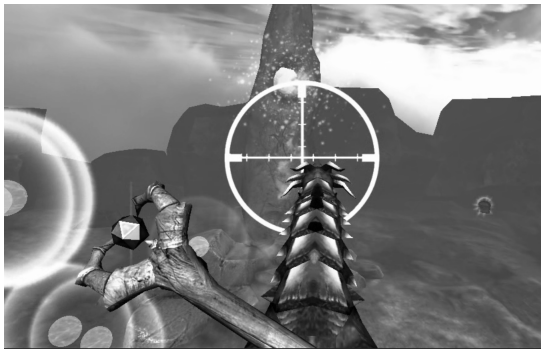


[Fig. 9] Experimental Results under Virtual Network Simulation.

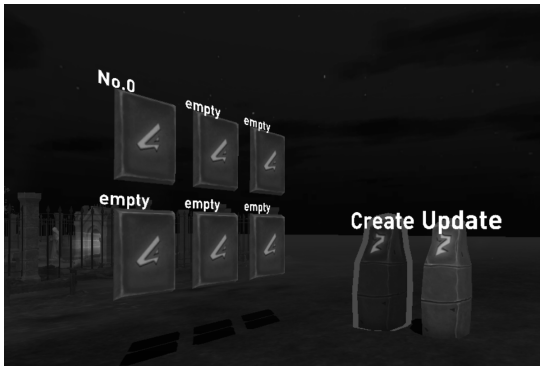
4.2. 다중 접속 온라인 게임 시뮬레이션

실험에 사용한 다중 접속 온라인 게임 “니드호그 라이더”는 유니티 엔진 기반으로 자체 제작한 온라인 FPS 게임이다. [Fig. 10]과 같이 온라인에서 사용자간에 대전하는 비행 슈팅 게임이다.

[Table 1]와 [Table 2]은 실험에 사용한 동적 오브젝트와 정적 오브젝트 메쉬의 크기를 정점과 삼각형 개수로 설명한다. [Fig. 11]는 자체 제작 게임 “니드호그 라이더”에서 사용한 통신 패킷들을 설명하며 크게 4가지로 나뉜다. “Move” 패킷은 모든 객체(Player, Bullet 등)들의 이동 정보를 가지고 있고 게임에서 가장 많이 발생되는 패킷이다. “Create” 패킷은 모든 객체(Player, Bullet 등)들의 생성 정보를 가지고 있다. “Destroy” 패킷은 모든 객체(Player, Bullet 등)들의 파괴 정보를 가지고 있다. “Hit” 패킷은 Bullet과 Player의 충돌체크 결과를 가진 패킷이며 패킷들 가운데 가장 생성이 적게 된다.



(a) Game Play



(b) Game Lobby

[Fig. 10] Screen Shots of ‘NidhoggRider’

[Table 1] Dynamic Objects

	Dragon	Player
No. of Vertices	5,678	6,270
No. of Triangles	4,798	9,854

[Table 2] Static Objects

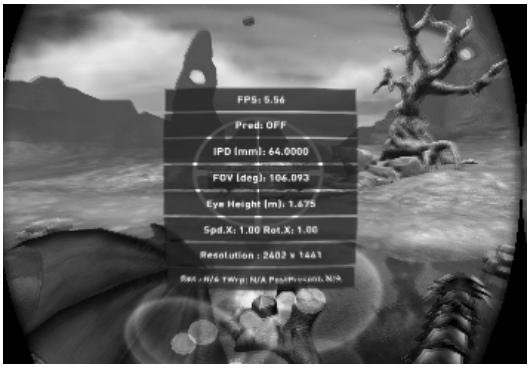
	Rock1	Rock2	Tree
No. of Vertices	653	288	5,007
No. of Triangles	1,243	478	8,558

Packet type	Packet generation probability
Move	High
Create	Middle
Destroy	Middle
Hit	Low

[Fig. 11] Packet Types of 'NidhoggRider'

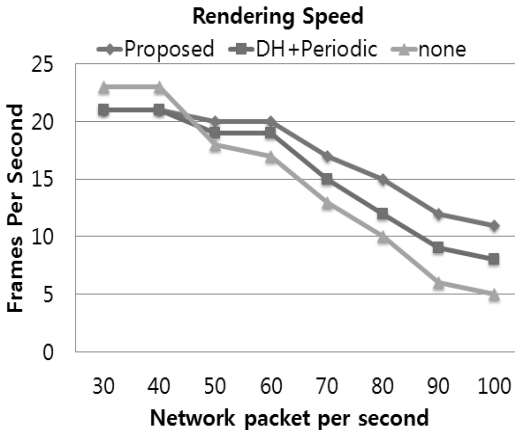
Packet type	Partial Packet Discard	Periodic Packet Discard
Move	O	O
Create	△	△
Destroy	△	X
Hit	X	X

[Fig. 12] Packet Discard Possibilities (O=all, △=partial, X=none)



[Fig. 13] Screen shot of FPS (Frames Per Second)

[Fig. 12]은 제안하는 기법에서 Partial Packet Discard[8] 기법과 Periodic Packet Discard[9] 기법을 적용할 때 패킷 폐기 가능성을 설명한다. Partial Packet Discard[8] 기법을 적용할 때 "Move" 패킷은 객체가 같을 경우 폐기가 가능하고, "Create" 패킷과 "Destroy" 패킷은 큐 안에 동일한 객체의 "Create"와 "Destroy" 패킷이 동시에 존재하는 경우 두 패킷 모두 폐기가 가능며, "Hit" 패킷의 경우는 어떠한 경우에도 폐기가 불가능하다. Periodic Packet Discard[9] 기법을 적용할 때 "Move" 패킷의 경우 객체의 이동 정보가 오래될 경우 신뢰성을 잃게 되므로 폐기가 가능하고, "Create" 패킷의 경우 게임 내의 이펙트나 파티클



[Fig. 14] Experimental Results under Real Multi-player Online Game Simulation.

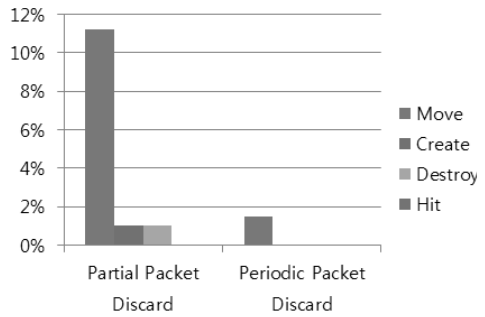
생성 명령이 일정 시간을 지날 경우 폐기가 가능하며, "Destroy"와 "Hit" 패킷은 폐기가 불가능하다.

시뮬레이션 환경은 [Fig. 11]의 패킷 생성 비율을 고려하여 4종류의 패킷이 임의로 생성된다. [Fig. 13]은 게임 시뮬레이션에서 렌더링 속도 FPS(Frames Per Second)를 캡처한 것이다. [Fig. 14]는 제안하는 패킷 폐기 기법을 사용하는 "Proposed", Drop Head와 Periodic Packet Discard 기법을 사용한 "DH+Periodic" 그리고 패킷 폐기를 하지 않는 "none"의 게임 렌더링 속도를 비교한다. 초기 패킷 생성이 적은 시점에서는 "none"의 렌더링 속도가 높으나 점차 패킷 성능이 많아지면 제안하는 기법의 게임 렌더링 속도가 향상된다. 또한 "DH+Periodic"의 경우 초기 패킷 생성이 적은 시점에서는 제안하는 기법과 비슷하나 패킷 생성량이 많아지면서 급격히 게임 렌더링 속도가 저하된다. 전체적으로 제안하는 기법은 패킷 폐기를 하지 않았을 때 보다 렌더링 속도가 평균 39%, 최대 120% 향상된다. 그리고 Drop Head와 Periodic Packet Discard 기법을 사용한 기법보다 평균 10%, 최대 38% 향상된다.

[Fig. 15]는 [Fig 14]의 실험에서 초당 패킷 생성률이 100일 때 제안하는 기법에서 Partial Packet Discard[8] 기법과 Periodic Packet Discard[9]기법의 패킷 폐기율을 보여준다. Partial Packet Discard[8]의 경우 대부분이 "Move"패킷을 폐기하였고 전체 "Move"패킷의 11%를 폐기하

였다. 또한 “Create”와 “Destroy” 패킷은 1%로 저조하였는데 객체를 바로 만들고 삭제하는 경우의 수가 적기 때문이다. 하지만 실제 다중 접속 온라인 게임에서는 접속을 했다가 바로 끊어버리는 경우가 많기 때문에 실제 환경에서는 더 좋은 성능을 보일것이라고 예측할 수 있다.

그에 비해 Periodic Packet Discard[9] 기법을 적용한 경우 폐기율이 전체적으로 저조하였는데 그 이유는 Partial Packet Discard[8] 기법에 의해서 상당 부분 폐기되었기 때문이다. 패킷 폐기 주기를 조정하면 패킷 폐기율이 향상될 것으로 예상된다.



[Fig. 15] Packet Discard Ratio (%)

5. 결 론

본 논문에서는 유니티 엔진 기반 다중 접속 온라인 게임에서 스레드 간의 통신 패킷 폐기기법을 제안하였다. 제안하는 기법은 네트워크 패킷을 큐로 관리하고 중복되는 패킷을 주기적으로 폐기하여 혼잡한 네트워크 상황에서의 빠른 패킷처리와 적은 지연시간을 가진다. 제안하는 기법을 활용할 경우에 스레드 통신 큐의 크기가 줄어들고 게임 렌더링 속도가 향상된다. 제안하는 기법은 자체 제작한 유니티 엔진 기반 다중 접속 온라인 게임의 다양한 패킷 발생 시뮬레이션을 통하여 렌더링 속도를 분석하고 효율성을 증명하였다.

제안하는 기법은 패킷 통신량이 많은 MMORPG 게임이나 비슷한 정보를 가진 패킷이 많은 AOS, FPS 게임에 적용한다면 상당히 높은 클라이언트의 성능 향상을 기대할 수 있다. 특히 클라이언트의 렌더링 속도가 네트워크 통신 성능에 의존하면 큰 성능 향상을 기대할 수 있다.

다만 제안하는 기법을 게임에 적용할 때 패킷의 폐기를 단순히 버퍼 안에 동일한 종류의 패킷으로 판별하여 폐기하지 않고 패킷의 신뢰성이나 게임 로직을 고려해야 한다.

REFERENCES

[1] KOCCA, “2014 White Paper on Korean Games”, pp.21, 2014.

[2] C. J. Lim, Won Dae Han, Jeong Yun Guen, “Educational Game Making-Tool Development using Unity3D Engine: Birth of Game”, Journal of Korea Game Society, Vol. 14, No. 1, pp.29~38, 2014.

[3] Jou-Hyoung Lee, Jae-Hong Jeon, Min Hong, “Implementation of Multi-Platform Game Application using Unity Game Engine”, Korean Society for Internet Information, Vol. 13, No. 2, pp133~134, 2012.

[4] Unity Korea, https://unity3d.com/kr/

[5] HyoungGu Lee, Ik Jae Jeon, “Design and Implementation of the NonTargeting AOS Online Game”, Journal of Korea Game Society, Vol. 14, No. 5, pp.25~34, 2014.

[6] Jong-Min Lim, Dong-Woo 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, 2014.

[7] HyoJoo Park and TaeYong Kim, “Traffic Analysis and Modeling for Network Games”, Journal of Korea Multimedia Society, Vol. 9, No. 5, pp.635~648, 2006.

[8] Myung-Hee Kim, Seung-Seop Park, “Packet discard technique to improve the performance of the Internet Protocol”, Korea Multimedia Society, Vol. 4, No. 2, pp.15~23, 2000.

[9] Sung-Hyung Lee, Hyun-Jin Lee, Jae-Ryong Cha, Jae-Hyun Kim, Dong-Won Kum, Hae-Hyeon Baek, Sang-heon Shin, Jehyun

Jun, “Periodic Packet Discard Policy for Frame Based Scheduler”, Korea Information and Communications Society, Vol. 38, No. 2, pp.97~104, 2013.

[10] S. Kim, T. Park, and C. Kim, “Mean value analysis of the waiting time for the drop-head buffer management,” IEICE Trans. on Commun., vol. E85-B, no. 9, pp. 1860-1862, Sep. 2002.

유 종 근(Yoo, Jong Kun)

한국산업기술대학교 게임공학과 전공 재학 중

관심분야 : 게임프로그래밍 , 서버프로그래밍

김 영 식(Youngsik Kim)

1993 연세대학교 컴퓨터과학과 학사
1995 연세대학교 컴퓨터과학과 석사
1999 연세대학교 컴퓨터과학과 박사
1999~2005 삼성전자 System LSI 책임연구원
2013 University of Pittsburgh 방문교수
2005~ 한국산업기술대학교 부교수

관심분야 : 게임기구조, 컴퓨터구조, 3차원 그래픽가속기, 임베디드 시스템 등