P2P와 UNET 이해

멀티플레이가 제공되면 사용자 간 관계의 긴밀도가 올라간다. 여러 가지 영향을 미친다. SNS와 게임 방송이 활성화된 요즘 게임 컨텐츠로서 중요하다.

서버 기반으로 멀티플레이 기능을 제공하는 방법이 이상적이긴 하나 경험이 축적되어 있지 않다면 상당히 어려운 면이 있다. 클라이언트 간 직접 통신을 통해 멀티 플레이 기능을 제공하는 것이 빠른 구현의 선택이 될 수 있다. 이를 P2P라 한다. P2P는 서버 처리를 거치지 않고 클라이언트들이 물리적으로 가깝다면 서버를 통하는 것보다 빠를 수 있다. 초기 FPS 게임들, 카드라이더 등 많은 게임들이 P2P 방식으로 성공했다.

P2P

개요

P2P는 peer to peer의 약자. 클라이언트 간의 통신으로 멀티 플레이를 가능하게 하는 구조이다. C/S (Client/Server) 방시과 대별되며 다양한 구성의 차이를 갖는다. 중요 선택은 다음과 같다.

- UDP vs. TCP
- Master (Host) Peer vs. Full P2P (대등한 피어들)

UDP vs. TCP

UDP가 TCP 보다 홀펀칭 가능성이 좀 더 높기 때문에 P2P의 경우 UDP를 사용하는 경우가 많다.

UDP 기반이라고 하더라도 신뢰성을 보장하는 프로토콜을 사용해야 할 필요가 있다. 신뢰성 없는 프로토콜은 퀘이크 시절부터 위치 동기화에만 주로 사용했다. 신뢰성 없는 프로토콜은 전송 빈도가 높은 정보에 적합하고 모바일의 경우 데이터 비용이 발생하므로 가능하다면 사용하지 않는 것이 좋아 보인다.

Master (호스트) 방식과 대등한 P2P

모든 클라이언트를 대등하게 구성하면 판정이 단일하게 이루어지지 않는다. 만약 게임에서 판정을 각자 할 수 없는 경우 클라이언트 하나를 마스터(호스트)로 정하여 중요 판정을 한 곳에서 하도록 한다. 이런 경우 결과 값의 전송 시간만큼 Slave 클라이언트들에서 표시까지 지연이 발생한다.

중요 이슈

P2P 방식을 사용할 경우 클라이언트 간의 연결을 사용한다. IPv4의 주소 공간이 부족하여 와이파이 장비를 포함하여 NAT (Network Address Translation)가 대부분 사용된다. 이럴 경우 공인 IP를 통한 클라이언트 간 연결이 불가능한데 UDP 포트 매핑을 통해 내부 IP와 외부 IP간 연결을 맺는 방법들이 제안 되었다. 이를 홀 펀칭이라 한다.

홀 펀칭

NAT 장비의 IP, 내부 장비의 IP와 외부 IP 간 UDP 연결을 맺을 수 있도록 해주는 기능이 NAT 장비들에 있다. 이를 자동으로 맺는 방법들이 있으며 기본 아이디어는 다음과 같다.

- 외부 장비와 내부 장비 간의 UDP 연결을 맺는다.
    - 이 때, NAT 장비에서는 <내부 IP, 내부 포트, 외부 IP, 외부 포트> 테이블을 생성한다.
- 다른 외부 장비에서 <외부 IP, 외부 포트>로 UDP 패킷이 오면 <내부 IP, 내부 포트>로 전송한다

보안 등을 위해 몇 가지 프로토콜이 있고 이들을 처리해 주는 기능이 Raknet에 구현되어 있다.

http://www.raknet.net/raknet/manual/natpunchthrough.html

표. 두 대의 NAT간 타잎에 따른 성공 실패

Router Type	***Full cone NAT***	***Address-Restricted cone NAT***	***Port-Restricted cone NAT***	***Symmetric NAT***
Full cone NAT	YES	YES	YES	YES
Address-Restricted cone NAT	YES	YES	YES	YES
Port-Restricted cone NAT	YES	YES	YES	NO
Symmetric NAT	YES	YES	NO	NO

릴레이

홀펀칭 기능을 사용하더라도 방화벽 뒤쪽에서 게임 플레이를 하는 경우 UDP 통신이 막히는 경우가 많다. UDP의 일부 프로토콜만 보내는 것을 허용하는 경우가 대부분으로 실제 게임 통신은 회사 같은 경우 불가능한 경우가 많다. 이럴 경우 TCP를 사용하거나 불가능한 경우 UDP를 통해 통신을 중계해줘야 하며 이 기능을 릴레이, 해당 기능을 하는 애플리케이션을 릴레이 서버라 한다.

릴레이는 P2P 통신 그룹 단위로 형성하고 빠르게 전송 하는 것이 가장 중요하다. 이 기능도 Raknet에 구현되어 있다.

패킷 크기

UDP에 기반한 프로토콜을 설계할 때 될 수 있으면 512 바이트 이내로 패킷을 만드는 것이 좋다. 네트워크로 전송 가능한 최대 크기가 이더넷은 1514이고 WAN의 경우 512 바이트 정도인 연결이 있어 이 보다 클 경우 잘라서 보내야 한다. 보통 나누고 합치기 기능 (Segmentation / Reassembly)은 구현 복잡도를 올리고 응답 속도에도 영향을 미친다.

UDP 기반 P2P 게임은 분산된 환경에서 필드 테스트를 꼭 거쳐야 하는 이유가 위와 같은 이유들 때문이다.

마스터 이전 (Migration)

2명 이상이 플레이 할 경우 마스터로 동작하던 클라이언트가 게임에서 나갈 경우 마스터를 변경해야 한다. 이를 이전 (MIgration)이라고 한다. 게임을 잠시 멈추고 특정 클라이언트를 마스터로 전환하는 과정과 상태 조정 과정을 거쳐야 한다. 이를 단순하게 하기 위해 모든 상태를 갖는 부마스터를 두는 방법이 많이 사용된다.

단선 처리

모바일 환경은 네트워크 단선이 많으므로 단선을 클라이언트 종료로 인식 하지 않아야 한다. 가장 쉬운 방법은 다시 접속하게 하는 것이다. 그 다음으로 쉬운 방법은 다시 접속 하는 과정을 클라이언트에서 자동으로 진행시키는 방법이다.

게임마다 다르겠지만 친절한 메세지 표시, 재대결 등 기획적인 방법으로 해결할 수도 있으므로 고려하여 결정해야 한다. 구현이 어려울 수 있으므로 너무 기술적인 부분에만 의존하지 말아야 한다.

UNET

UNET은 오픈소스 프로젝트로 진행되고 있다. 아래 링크에서 소스를 볼 수 있다.

https://bitbucket.org/Unity-Technologies/networking

몇 가지 인터넷 자료가 괜찮은 것들이 있어 읽고 요약하면서 핵심을 정리하고자 한다.

개요

참고 자료 [1]로 대략의 개념과 용도를 살펴본다. 2015년 문서이므로 현재와 차이가 있을 수 있다는 점은 유념해야 한다.

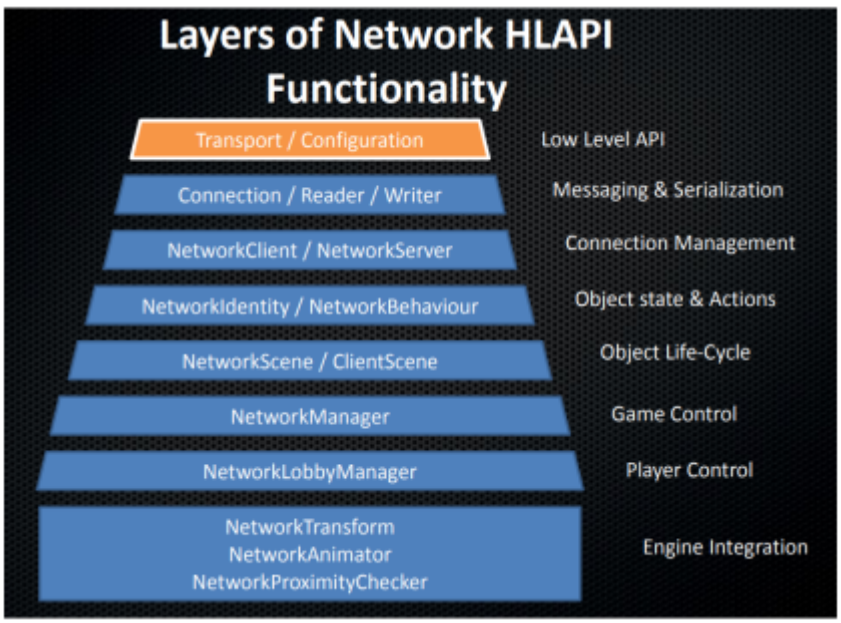
- Raknet에서 시작하고 공존. 현재는 제거된 것으로 보임
- P2P 게임에 적합
  - MMO 용 아님
- C# HLAPI와 Transport Layer로 구성
- Transport Layer
  - UDP
  - C++로 구현
  - Reliable / Unrealiable 지원

위 정도가 개발의 기본 철학과 구조를 보여준다.

API는 HLAPI (High Level API)와 LLAPI (Low Level API)로 나눈다.

- HLAPI
  - C# extension DLL
  - UnityEngine.Networking namespace
  - 메세지 핸들러
  - 일반적인 serialization
  - 분산 오브젝트 관리
  - 상태 동기화
  - Server, Client, Connection 등의 클래스
- Online Services
  - 유니티 클라우드에서 서비스 제공
  - Relay Server
    - NAT와 방화벽 이슈를 해결
  - 매치메이킹
  - 클라이언트
    - Web service interface
    - C# wrapper를 제공

매치메이킹과 릴레이를 클라우드에서 제공하고 HLAPI를 통해 연결을 맺고 동기화 하고 메세지를 처리한다. 여기까지 설명으로 보면 괜찮은 방향을 갖고 있다.



HLAPI는 레이어 구조를 갖고 있다. LLAPI부터 시작하여 엔진 연동까지 구성된다.

Dedicated Server (데디케이트 서버)

클라이언트에서 렌더링을 제거하고 게임 시뮬레이션 서버로 사용하는 서버를 데디(케이트) 서버라고 한다. 언리얼과 같은 엔진에서 FPS 게임의 서버로 제공한 경우들이 있고 보통 서버 당 지원하는 동접이 작다.

UNET에서 데디 서버를 지원하기는 하나 P2P로 가능하다면 선택하지 않는 것이 좋다. 동접 확장성이 좋지 않은 경우가 많고 관리 부담이 큰 편이다. UDP를 사용하는 서버를 IDC에 두는 것이 보안 상으로 이슈를 야기할 가능성도 크다.

프로토콜 설계

인터넷 검색에서 UNET 관련 이슈들이 2017년까지 올라온다. 특히, 이동 동기화가 안 좋다거나 릴레이가 느리다거나 하는 항목들이 있다. 이를 우회하려면 패킷 크기를 작게 하고 입력 위주로 동기화하고 중요 결과만 동기화 하는 노력이 필요해 보인다. 특히, NetworkTransform, NetworkAnimator와 같이 무거운 동기화들은 게임에 따라 적절하게 별도로 구현하는 걸 권한 내용들이 꽤 있다.

게임 특성에 맞게 프로토콜을 선택한다.

- 입력 동기화 만으로 가능한가?
- 판정 동기화가 필요한가?
- 스폰 동기화가 필요한가?
- 이동 동기화가 필요한가?
- 연출 동기화가 필요한가?

이동 동기화가 필요한 게임이라면 동기화의 모든 문제를 만나게 된다. UDP의 특성을 잘 활용하여 주기적인 업데이트, 방향과 속도의 활용을 통해 최대한 부드럽게 만들어야 한다. 이동 동기화가 필요한 게임은 보통 판정도 동기화 해야 한다. 왜냐하면 양 쪽 클라이언트 간의 위치가 달라지는 걸 피할 수 없기 때문이다.

중요한 고려 사항 중 하나는 PvP vs. PvE에 따라 달라진다. PvE는 협동 플레이인 경우가 대부분으로 동기화가 맞지 않더라도 게임에 참여한 전원에게 유리한 방향으로 판정이 이루어진다면 불만이 없게 된다.

턴 방식인가 아닌가도 중요하다. 턴 방식이라면 입력 동기화 만으로 각자 완전하게 시뮬레이션이 가능하다.

전략 게임 (RTS) 의 동기화

스타크래프트와 같은 전략 게임은 수 많은 유닛을 포함한다. 개별 유닛의 이동과 행동을 동기화 하는 방식은 트래픽이 너무 많이 발생하기 때문에 입력 동기화 만으로 처리한다. 입력을 모든 클라이언트(피어)들에서 받으면 다음 단계로 진행한다.

이와 같은 아이디어를 기반으로 입력을 모두 받은 후 각자 처리를 진행하는 방식이 가장 간결한 동기화 방식이다. 에이지 오브 엠파이어의 네트워크 구현 발표로 대중화 되었다.

핵심 아이디어들은 다음과 같다.

- 200ms 마다 새로운 턴이 시작된다.
- 턴 중에 입력을 받아 들인다
- 턴이 끝날 때 전송 한다.
- 모든 사용자의 턴 입력이 오면 처리를 진행한다

    - 여기서 대기 할 수 있으므로 2턴 이전의 명령을 실행한다

실행 예제.

```
턴 1000 : 플레이어 A 스킬 3, 플레이어 B 스킬 5 입력

턴 1001 : 몬스터 B 스킬로 플레이어 B 공격 입력

턴 1002 : 실행 --> 플레이어 A 스킬 3, 플레이어 B 스킬 5

턴 1003 : 몬스터 B 스킬로 플레이어 B 공격 실행
```

위에서 1002, 1003에서도 입력은 처리한다. 턴 1001의 몬스터 입력은 마스터 서버에서 입력을 생성한다.

턴 방식 게임일 경우는 특별히 걱정할 것이 없다. 이미 턴으로 진행되기 때문이다. 만약, 더 빠른 입력이 필요한 게임이라면 턴 시간을 줄여야 한다. 그렇게 될 경우 네트워크 지연으로 인해 입력을 건너뛰는 경우도 생길 수 있다. 핵심은 **모든** 플레이어의 **입력**을 받아서 **이전 턴**을 실행하는 것이다.

원칙들

- 패킷 크기와 전송 횟수는 최소화 한다.
    - 입력만으로 동기화 가능하면 그렇게 한다.
    - 각자 시뮬레이션으로 처리 가능하면 그렇게 한다.
- 동기화 시간을 벌 수 있는 지점을 확보한다
    - 애니메이션이나 사후 연출 등으로 시간을 확보한다
    - 동기화 지연 시간은 미리 정하고 설계한다
    - 부드러운 느낌이 실제 지연보다 더 중요하다
- 동기화 지연이 사용자에게 손해가 되지 않도록 기획한다
    - 랙이 있어도 죽거나 손해가 없다면 불만이 덜 하다
    - RPG와 같이 불가능한 경우도 있다.
    - 될 수 있으면 최대한 그렇게 한다.

서비스 구성

Unity의 매치 메이킹과 릴레이 서버를 사용할 것인 지, Raknet의 홀 펀칭을 사용할 것인지 결정해야 한다. 게임이 무겁지 않다면 Unity의 릴레이 서버만 사용하거나 자체 릴레이 서버를 두는 정도로만 구현하는 것이 적절한 선택으로 보인다.

홀펀칭이 필요한 정도로 큰 게임이라면 클라이언트 / 서버 방식을 검토해야 할 것으로 보인다. MORPG 정도라던가, RTS 게임이라면 그 정도의 투자가 있어야 안정적인 서비스가 가능해 보인다. P2P의 게임 플레이 검증 이슈 외에 모바일에서는 단선 시 재진입도 중요하기 때문이다.

참고 자료

[1] unite2015 UNET 프리젠테이션

http://japan.unity3d.com/unite/unite2015/files/DAY2_1700_room1_Sean.pdf

- UNET 개요
- HLAPI와 LLAPI 구분 설명

[2] Real-time Unity Multiplayer Server Implementation

https://www.theseus.fi/bitstream/handle/10024/136944/Ahde_Jani.pdf?sequence=1

- UNET을 사용한 멀티플레이 구현 "학사" 논문

[3] Unity Asset : Tanks Multiplayer

https://www.rebound-games.com/products/tanks-multiplayer/

- UNET 또는 PUN (Photon Unity Networking) 선택 가능한 구현
- 무엇이 가능한 지 이해하는 정도로 활용

[4] 에이지 오브 엠파이어 동기화

https://zoo.cs.yale.edu/classes/cs538/readings/papers/terrano_1500arch.pdf

- 1500 archers on 28.8
- 오래된 RTS 동기화의 고전 문서

[5] UNET 튜토리얼

https://www.youtube.com/watch?v=ovUnNJlREIk&list=PLwyZdDTyvucw5JhBMJxFwsYc1EbQYxr0G&index=1