Multiplayer Game Development

305900
2007년 가을학기
12/6/2007
박경신

Multiplayer Game

- 최근 게임 시장 동향은 멀티-플레이어 옵션이 기본적으로 따라옴 – 그렇지 않으면, $10 미만의 가격대임
- 멀티 플레이어 게임의 두 가지 종류:
  - 하나의 기계/콘솔에 둘 또는 그 이상의 플레이어가 게임을 할 수 있음
  - 분리된 네트워크로 연결된 기계에 둘 또는 그 이상의 플레이어가 게임을 할 수 있음
- 멀티 플레이어 게임에서 고려해야 할 점 두 가지 종류:
  - Gameplay issue – 재미있는 멀티-플레이어 경험을 주는 이슈
  - Game development issue – 소프트웨어 개발 이슈

Gameplay Issues

- 멀티플레이어 옵션을 지원하는 게임을 설계하는 것은 개발 후 간단한 추가 (adds-on)로 되지 않음
- 설계 단계에 있어서 일찍 고려되어야 함
- 왜냐면, 멀티플레이어는 그려야 할 폴리곤 개수를 증가시킬 수 있고 스크린을 분할해야 하기 때문에..
- 수직 vs. 수평 분할 윈도우 - 적이 어디에서 나타날지에 기반하여, 자연적으로 수평이 수직보다 중요함
- Co-operative (협동) vs. Melee (격투)
- Co-op:
  - 게임에서 co-op이 일반적인 스토리라인을 따라서 플레이 하겠는가? (Halo 같이)
  - 또는, 특정 멀티플레이어 경기장에서만 일어나는가? (Quake같이)
  - 만약 우리 중에 한 명이 죽는다면 무슨 일이 일어나는가? E.g. Halo에서는 아무도 플레이어의 파트너를 총 쏘지 않고 있으면 소생시킴
  - 만약 우리 둘 다 죽는다면 무슨 일이 일어나는가? Halo에서는 그 레벨로부터 다시 시작함

Gameplay Issues

- 네트워크 게임에서 같이 플레이 할 수 있는 다른 플레이어를 어떻게 찾는가? 짝짓기 서비스
- 네트워크 게임에서 서로를 어떻게 찾는가? 레이다
- 게임은 영속적 (Persistent) 인가? Networked Quake vs. Ultima Online / Everquest
- 동시에 플레이할 수 있는 플레이어들을 얼마나 많이 지원하는가?
- 멀티 플레이어 세션에서 NPCs를 두어야 하는가? 이 때, NPCs는 나의 친구인가 적인가?

E.g. Gauntlet

- 3인칭 관점 – 같은 스크린 안에 모든 사람들이 존재
- 게임플레이는 협동적 (co-operative)
- 플레이어가 다른 방향으로 간다면 무슨 일이 일어나는가?
- 카메라는 제한지점까지 확대

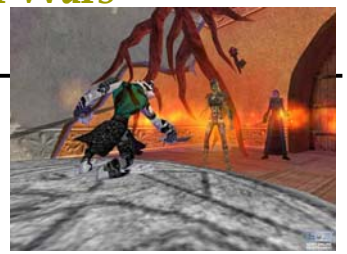
E.g. Quake / Jedi Outcast / Academy

- 1인 플레이어 전용의 전체 스크린
- 네트워크상으로 협동적 (co-op) & 격투 (melee) 플레이가 가능
- 도전 (challenge):
  - 네트워크 지연 (latency)
  - 매우 많은 플레이어와 NPCs를 지원
  - 사용자 지정 게임 캐릭터를 위한 mods 지원

E.g. Halo

- 분할 윈도우로 Co-op/melee와 네트워크 게임플레이에 사용
- Challenges:
  - 해상도 (screen resolution) 제한문제
  - 폴리곤 개수
  - 네트워크 지연 (network latency)
  - 매우 많은 플레이어와 NPCs를 지원

E.g. Ultima / Everquest / Star Wars Galaxies



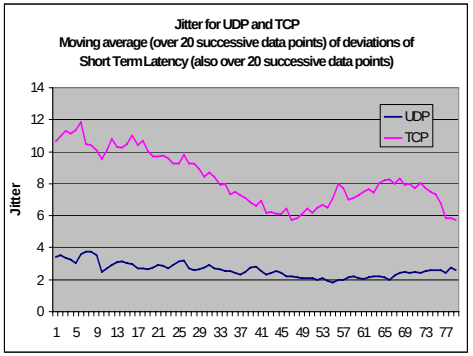
- 1인용 게임모드를 지원하지 않음
- 전체 스크린을 1인 플레이어에게 제공
- 모든 플레이어는 네트워크 상에 있음
- 방대한 스케일의 영속적인 RPG
- 끝이 없는 플레이
- 플레이들간의 사회적 상호작용을 통하여 콘텐츠가 제공되어 더욱 내용이 풍부해짐과 다시 플레이하기 기능이 향상
- 몇 수십 년간 백명 또는 천명 이상의 플레이어를 관리할 수 있어야 함
- 만약 플레이어가 자거나 일하러 갔을 땐 무슨 일이 일어나는가? 누가 그 플레이어의 성을 지켜줄 것인가?
- 일정하게 작동하는 시뮬레이션환경을 제공해주는 신뢰성 있는 데이타베이스 사용
- 플레이어 수의 증가에 따른 데이타베이스 이동 문제
- 기존의 서비스를 혼란시키지 않으면서 시간이 지남에 따라 어떻게 새로운 게임 플레이 기능을 편입시킬 수 있는가?

Fundamental Networking Issues

- TCP/IP Protocols
  - TCP – Connection oriented, Completely reliable
    - 데이타는 연결된 스트림으로 도착. 데이터가 스트림으로 오기때문에, 데이터의 어느 바이트가 언제 도착하는지 보장할 수 없음. 메시지를 만들기 위해 사용자가 바이트를 충분히 수집해야 함.
  - UDP – Non-connection oriented, Unreliable
    - 데이터는 개별적인 패킷(packets)/메시지 (messages)로 옴. 각 패킷은 정확함. 그러나, 패킷의 도착 순서는 보장할 수 없음.
- Bandwidth - Bits per second 단위로 측정
- Latency - 가장 큰 문제!
  - 일반적으로 거리, 라우터와 end-system의 데이터 오버플로우, 패킷 로스 (packet loss)에 의해 생김
  - 미국에서 Roundtrip times은 미국 내까지 (TransUS) 50ms; 대서양을 넘어서 (TransAtlantic) 120ms; 태평양을 넘어서 (TransPacific) 200ms
  - Milliseconds 단위로 측정
  - TCP는 데이터 스트림이 정확한지 확인하는 acknowledgement를 요하기 때문에 TCP의 Latency가 UDP보다 큼
  - 매우 협동적인 상호작용적 응용프로그램은 적어도 최소 200 ms roundtrip latency 이하를 유지해야 함

Fundamental Networking Issues

- Jitter
  - Variation of latency
  - 플레이어들이 개체의 움직임의 궤적을 예측하기 어렵기 때문에 Jitter는 latency 보다 더욱 거슬리는 존재임
  - Jitter는 TCP에서 UDP보다 훨씬 더 많음
  - TCP에서 일반적으로 패킷 로스 (packet loss)를 회복 때문에 발생
  - 네트워크 traffic shaping을 사용하여 Jitter를 부드럽게 하거나 아바타의 위치를 평균하여 움직임을 부드럽게 만듬



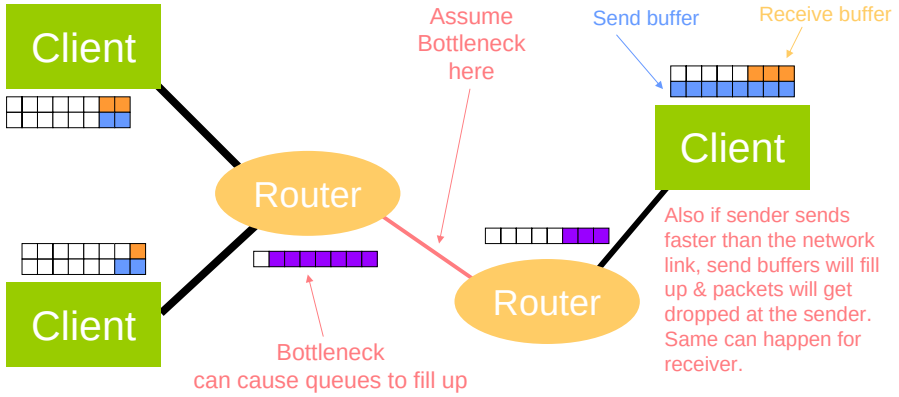
Fundamental Networking Issues

- Unicast, broadcast, multicast
  - Broadcast는 동시에 여러 수신자들에게 메시지를 보냄
  - Multicast는 1개 메시지를 보내고 라우터가 사본을 만들어서 (UDP 기반으로) 메시지를 보냄
  - 문제는 multicast가 인터넷에 네트워크를 넘치게 만들기 때문에 대부분의 라우터에서 multicast를 지원하지 않음
- Firewalls
  - 일반적으로 네트워크 응용 프로그램은 알려진 포트(port) 번호 사용
  - Firewalls은 컴퓨터로 들어오는 원치 않는 네트워크 트래픽을 막음
  - 그러나 이 firewall가 네트워크 게임 응용프로그램도 종종 막는 경우가 있음
  - 따라서 firewall 가 열릴 수 있게 게임은 포트 번호를 알려줘야 함
  - 이는 불행히도 해커들에게 취약한 문제점을 만들 수 있음

Networking Issues that can Ruin a Game Experience

Routers와 end-systems은 네트워크 혼잡 (congestion)할 시에 데이터를 저장할 네트워크 버퍼 (즉, 큐)를 가지고 있음.

Congestion은 일반적으로 네트워크 트래픽(traffic) 때문에 발생됨 – 라우터가 데이터를 전달할 수 있는 양을 넘칠 때



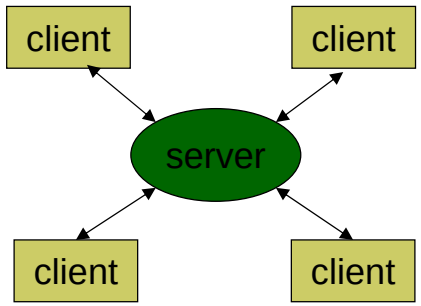
Fundamental Networking Issues

- 네트워크 버퍼가 가득 차면, 패킷이 하락
  - TCP에게 미치는 영향
    - Latency가 증가됨 – 왜냐면 확실한 신뢰성있는 배달을 보장하기 위해 TCP는 패킷을 다시 보내기 때문에
    - Latency가 증가됨- 왜냐면 네트워크 혼잡을 줄이기 위해 TCP가 transmission rate를 줄이기 때문에
  - UDP에게 미치는 영향
    - 간단히 데이터를 잃어버림
- 무엇이 가장 좋은 sending rate인지 쉽게 알 수 있는 방법이 있는가?
  - 불행히도 없음 – 인터넷은 Quality of Service가 없음 – 가장 최선의 방법은 네트워크가 전혀 느려지지 않을 것이다라고 가정하고, 또 느린 플레이어에게 최소한으로 받아들일 수 있는 threshold로 맞추기 위해 동적으로 sending rate을 조절해야 함
  - 이런 sending rate 조절은 오디오/비디오 스트리밍에서는 흔히 사용되는 방법임
- 모든 클라이언트를 위해서는, 목표지점에 혼잡을 피하기 위해 네트워크에 들어오는 패킷을 최대한 빠른 속도로 처리함

Connectivity Models

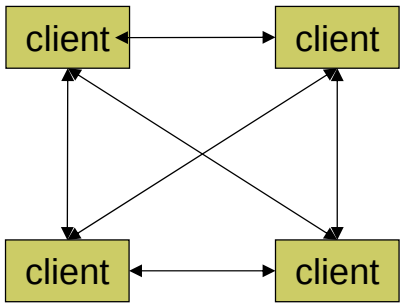
- Shared Centralized
  - 클라이언트는 중앙집중 서버에 연결 – 구현이 쉽고 모든 클라언트들에게 항상성을 유지하는 게 쉬움 – 하지만 서버에 로드(load)가 심하고, 그 때문에 동시에 접속한 클라이언트의 수가 제한적임
- Replicated Homogeneous
  - 중앙집중 통제없이 multicast로 클라이언트들이 연결 – 다수의 클라이언트에 가장 적합한 해법이나, multicast는 전체 인터넷에서 지원되진 않음
- Shared distributed with peer-to-peer updates
  - 클라이언트들 peer 간에 서로 연결 – 종종 중앙의 mediator가 peer들의 연결을 시작함
- Shared distributed using client-server subgrouping
  - AOI (Area of Interest) 관리를 위한 subgrouping – 매우 큰 게임세상을 위한 가장 확장성있는 해법임 (예, Ultima Online, etc)

Connectivity Model: Centralized



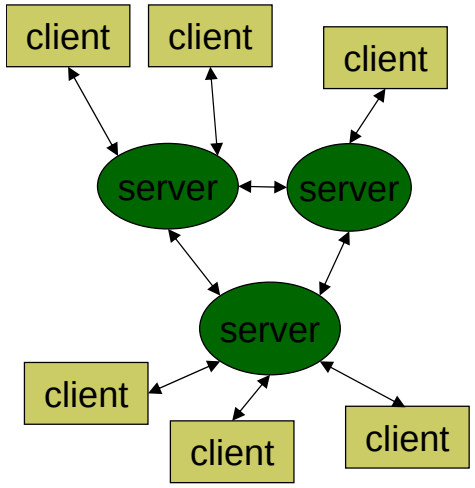
- 클라이언트-서버 모델
- 1 서버 컴퓨터가 최신 데이터를 모두 수집해서 모든 클라이언트들에게 전달함
- 단순한 구조이며, 데이터베이스 유지가 용이함 (useful for compression & admin tasks)
- 확장성이 없고, 중앙 서버가 일이 집중되므로 bottleneck이 생김

Connectivity Model: Distributed



- Peer-to-peer 모델
- 각 클라이언트가 자신의 고유 데이터베이스를 관리
- 최신 갱신정보는 다른 클라이언트들에게 전달
- 모든 연결을 관리하는 것이 어려움
- 확장성이 없고, 네트워크의 양이 많아지므로 네트워크가 bottleneck이 됨

Connectivity Model: Hybrid



- 멀티-플레이어를 위한 멀티-서버를 가진 클라이언트-서버 모델
- 게임에서 많이 사용되는 모델

Extending game loop to handle networking

- Multi-threaded 게임 루프:
  - Input Loop Thread
  - Network Receive Loop Thread
  - Compute Loop Thread (put network send calls here)
  - Draw Loop Thread
  - Sound Loop Thread

Implementing a Network Receive Loop

- Socket 프로그래밍 (UNIX, Windows)
- In a thread:

```
While(1) {
    Use blocking select() call to determine which incoming socket has data to
      be read.
    Read data from ready socket.
}
```

- Blocking을 호출하지 않으면, while loop가 계속해서 돌게 되므로 CPU를 잡아먹게 됨
- 네트워크 게임프로그램에서는 이 부분을 메인 루프에 한 번 불러서 모든 메시지를 점검하고 읽어들임
- 또는 메인 루프에서 사용할 수 있도록 별도의 스레드에서 모든 네트워크 메시지를 수집하도록 구현함
- 네트워크 가상현실 라이브러리 Quanta API
  www.evl.uic.edu/cavern/quanta

Network Game Management

- 만약 멀티-플레이어 게임이 대용량 데이터 파일 (예, 3차원 모델)을 받아야 한다면, 웹 브라우저에서 이미지 파일을 캐쉬(cache)해서 사용하듯이 로컬 드라이브에 캐쉬할 것
- 네트워크 엔티티:
  - 아바타 (Avatar)
    - 몸체의 위치와 방향, 머리의 방향, 미리 저장된 제스쳐
    - 각 클라이언트별로 로딩할 몸체의 부분의 리스트
  - (키 눌림이나 버튼 눌림 말고) 완전한 상태 데이터를 보낼 것
  - 플레이어의 위치와 같이, 일반적으로 자주 보내는 데이터는 UDP를 사용할 것
  - UDP를 사용할 땐, 1K bytes이하를 유지할 것 – 그렇치 않으면 도착을 보장할 수 없음
  - 여러 개의 데이커를 하나로 묶어서 보낼 것 – 한번에 하나씩 보내지 말 것
  - UDP는 패킷이 도착하는 순서를 보장할 수 없기 때문에, 메시지 안에 패킷 번호를 넣어서 보낼 것

Network Game Management

- Bandwidth requirement 예측
  - 게임 루프에 각각의 N 사이클 별로 얼마나 많은 bytes의 데이터가 보내지고 받아졌는지, 얼만큼의 시간이 걸렸는지를 계속 주시함
  - Total_bytes/elapsed_time를 계산하여 bandwidth 요구정도를 예측하는데 사용함 – 이 값을 최다수의 플레이어 수만큼 곱하여 upperbound 예측값을 얻음
  - 너의 게임이 실행 시에 얼마나 네트워크를 사용하고 있는지 알아내기 위해 send/receive rate을 출력해 볼 것
- Bandwidth requirement 를 줄임
  - Dead Reckoning을 사용 – (매 사이클보다는) 매 N 루프 사이클마다 위치, 속도, 가속도 정보를 보냄
  - 데이터가 바뀌었을 때 정보를 보냄

How to Implement someone firing a bullet

- 만약 총을 쏘게 되는 경우, "BULLET position velocity bullet_type" 같은 메시지를 broadcast 함
- 그럼 클라이언트들이 이 메시지를 받아서 각자 게임상에서 다른 총알 같은 관리해야 할 에이전트/엔티티 리스트에 추가함 – 즉, 어떻게 총알이 날아가는지 클라이언트 각자가 계산함
- 각 클라이언트는 총알이 클라이언트와 충돌했는지를 점검하고, 이 결과를 다른 모든 클라이언트들에게 알려줌 – 즉, 각 클라이언트는 이것이 죽었는지를 결정하는 주체임

Handling Late Joiners

- 게임의 한 세션에서 플레이어가 가입하는데, late joiner는 메인 서버에게 처음 연결될 때 가입함. 이 때 서버는 다른 클라이언트들에게 이 late joiner를 (IP 주소를 알려줌으로써) 소개함
- 각 클라이언트는 각자 이 새로운 joiner에 대한 자료구조 엔티티를 생성함. 각 클라이언트는 이 late joiner에게 연결함. Late joiner는 모든 클라이언트를 자료구조에 엔티티를 생성함
- Late joiner는 자신에 대한 데이터를 모두에게 보냄 (특히, 이 joiner의 아바타를 구성하는 몸체 부분 같은 데이터) 모든 클라이언트는 이 joiner에게 자신에 대한 데이터를 보냄
- 이러한 점 때문에 버튼 눌림이 아닌 완벽한 상태 정보를 보내야 할 필요가 있는 것임
- 게임에서 late joiner에게 불완전한 정보를 사용해서 일단 게임 플레이를 시작하도록 하거나 (예를 들어, 만약 머리가 없다면 일시적인 머리를 제공함), 또는 모든 클라이언트와 late joiner가 완전히 동기화될 때까지 기다린 후 게임 플레이를 하게 함
- Handling zombies
  - 만약 누군가가 게임에서 연결을 끊으면 아바타 리스트에서 이 플레이어를 지워야 함
  - UDP경우 connectionless이기 때문에 누군가가 연결이 끊어졌는지 알 수 없음 – 그래서, 모든 피어들이 세션에 연결은 TCP로 하도록 함. 그래서 연결이 끊어졌을 때도 이를 알 수 있도록 함