Networking

Patrick Cardwell
James Lowrey
Blaine Morbitzer

Overview

- Transmission protocol
  - UDP, TCP
- Game Networking Architecture
- Server/Client Interactions
  - Client-side prediction, Interpolation, Lag Compensation
- Backend As A Service
- Unity Demo
- QuakeWorld Overview
- Quake demo

Protocol

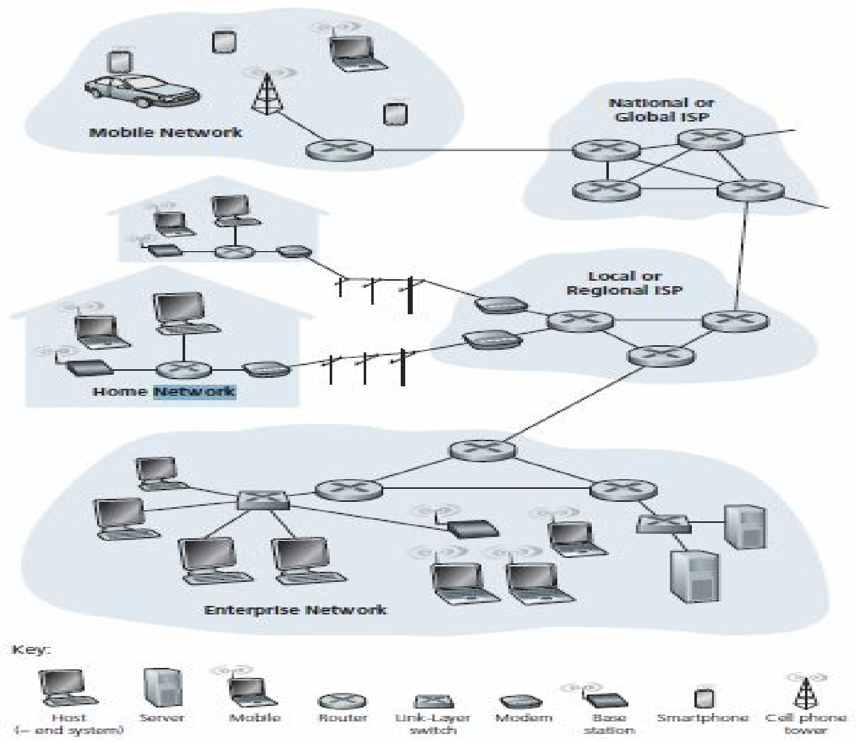
Definition: “Defines the format and the order of message exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event” - Computer Networking a Top Down Approach

Human Example: Ordering food at a restaurant

The Internet

Internet Protocol Stack

Application
Transport
Network
Link
Physical



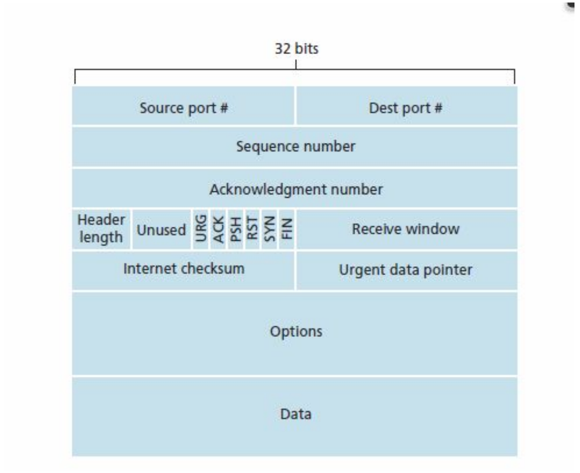
TCP - Transmission Control Protocol

- Connection oriented stream over an IP network: Reliable and Ordered
  - Guarantees data arrives in the order it was written
    - Uses acknowledgement packets sent back to the sender and automatic retransmission
  - Streams of data: TCP automatically splits data into packets and sends across network
  - Treats communication like writing and saving a file from one computer to another
  - Won’t send too much to the receiver
  - Sender will slow down if going too fast
- Backbone for almost everything you do online: web browsing, IRC, email, etc

TCP
ARE YOU GETTING ALL OF THIS?
NO, SLOW DOWN SO I CAN SAVE ACCURATELY!
SKULLBOX.NET

TCP continued

- Establishing a connection
  - Three way handshake
  - Takes 1 round-trip time (RTT)
- Segment Structure

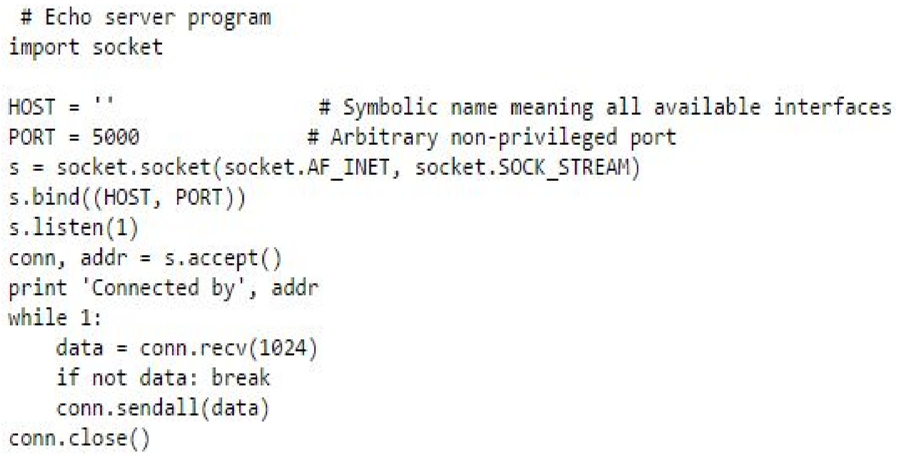


Definitions

- **Network Socket**: Endpoint of inter-process communication across a network
    - Comprised of local IP, port number, and transport protocol (UDP, TCP, raw IP)
- **Port**: Endpoint of communication in OS. ID’s specific process or service
    - 80: HTTP --- 21: FTP --- 194: IRC --- 443: HTTPS
- **Socket API**: APIs (usually by OS) that allows apps to control/use network sockets
- **Socket Address**: Combo of IP and port number

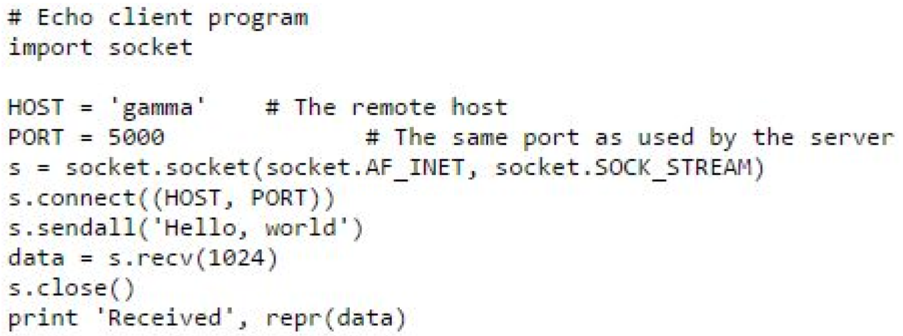
Sample TCP Python Code

Server Code



```
 # Echo server program
import socket

HOST = ''                 # Symbolic name meaning all available interfaces
PORT = 5000              # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.sendall(data)
conn.close()
```

Client Code



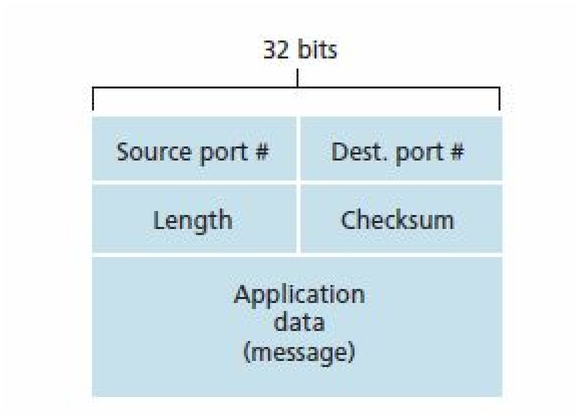
```
# Echo client program
import socket

HOST = 'gamma'    # The remote host
PORT = 5000              # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.sendall('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)
```

UDP - User Datagram Protocol

- Connection-less protocol.
  - Guarantees a given packet will arrive in whole or not at all
  - Packets can be received out of order with loss of packet information
    - Generally 1-5% loss and in order
  - Sends and receives packets directly: very thin layer over IP
  - Unreliable data transfer
  - No handshaking
  - Sent as fast as desired
- Used for real-time communication-small packet loss preferable to slow downs

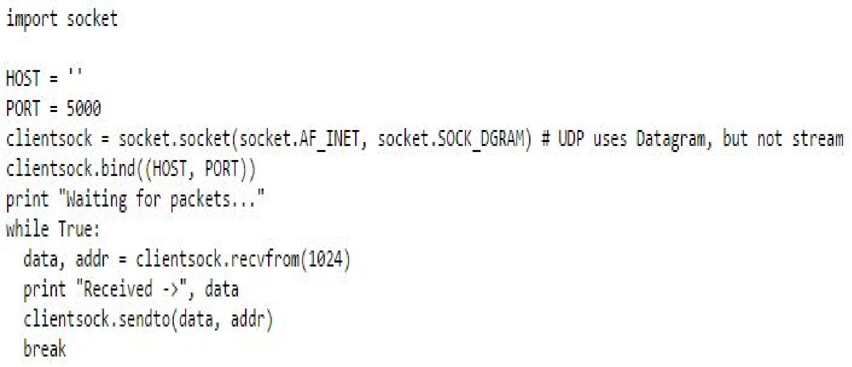
UDP continued

- Checksum
  - For error detection
  - 1's complement sum of 16 bit words
- Segment Structure
  - small segment header



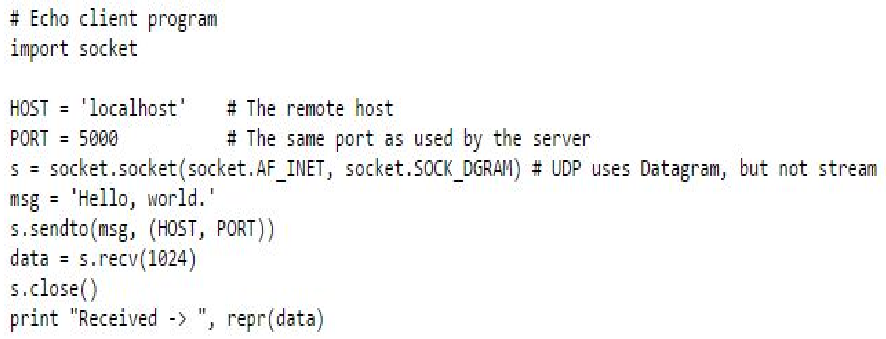
Sample UDP Python Code

Server Code



```
import socket

HOST = ''
PORT = 5000
clientsock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP uses Datagram, but not stream
clientsock.bind((HOST, PORT))
print "Waiting for packets..."
while True:
  data, addr = clientsock.recvfrom(1024)
  print "Received ->", data
  clientsock.sendto(data, addr)
  break
```

Client Code



```
# Echo client program
import socket

HOST = 'localhost'    # The remote host
PORT = 5000           # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP uses Datagram, but not stream
msg = 'Hello, world.'
s.sendto(msg, (HOST, PORT))
data = s.recv(1024)
s.close()
print "Received -> ", repr(data)
```

TCP vs UDP

What should you choose for your game state?

It depends...

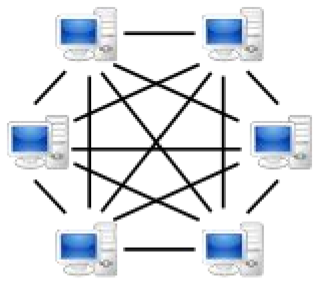
Games where timing is paramount and losing a few packets is acceptable then you should use UDP.

Examples: Counter Strike, League of Legends, etc.

Games where packets need to be reliably sent you should (probably) use TCP.

Examples: World of Warcraft, online poker, etc.

Game Network Architecture



Peer to Peer: direct connection and host/client

- First type of game networking
- Common today in bluetooth, local Wifi mobile games, and RTS

Pros: No cost to maintain servers, play not dependent on few servers, scales well

Cons: Difficult to implement, prevent cheating, maintain security, client limitations, Latency matches slowest peer

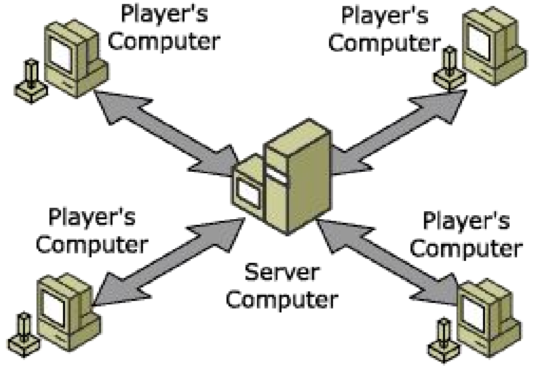
Game Network Architecture

Server/Client: single server that is responsible for running the main game logic

- First developed for Quake

Pros: Easier to implement, can scale well, better security, lower latency, game not affected by one poor client connection

Cons: Money, fewer servers

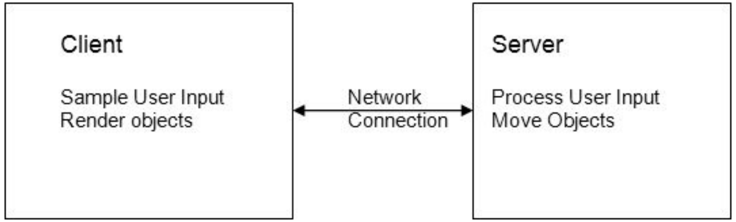


Server/Client

Client: Sends input, receives server packets, renders scene

Server: Read and execute input, simulate game world, send updates to clients

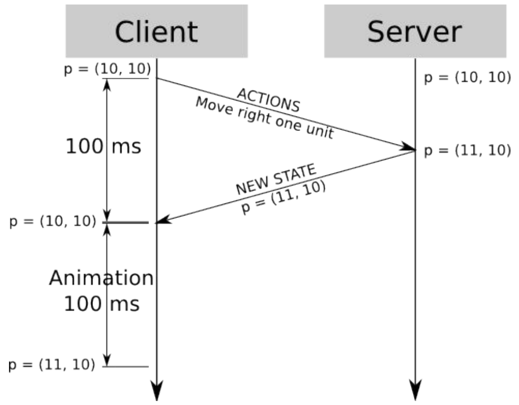
Lag!

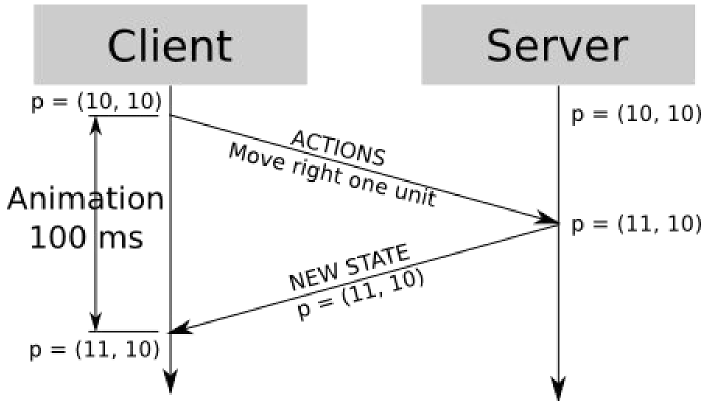


Client-Side Prediction

Naive: Client sends inputs, server processes & sends updated state, client moves

Client-Side Prediction: Send the input and start rendering the outcome of that inputs as if they had succeeded. Does not wait for authorization.



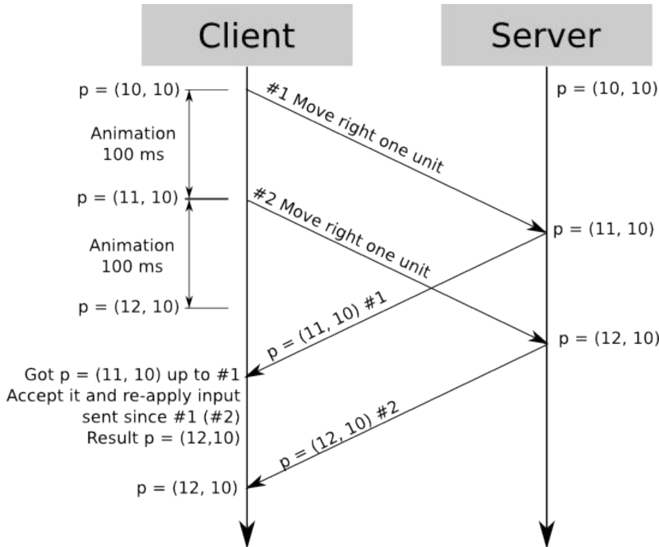


Client-Side Prediction

Synchronization: Client moves, server processes & sends response, Client receives old response and teleports back in time

Key point: Client is in present, Server is in the past (and authoritative)!

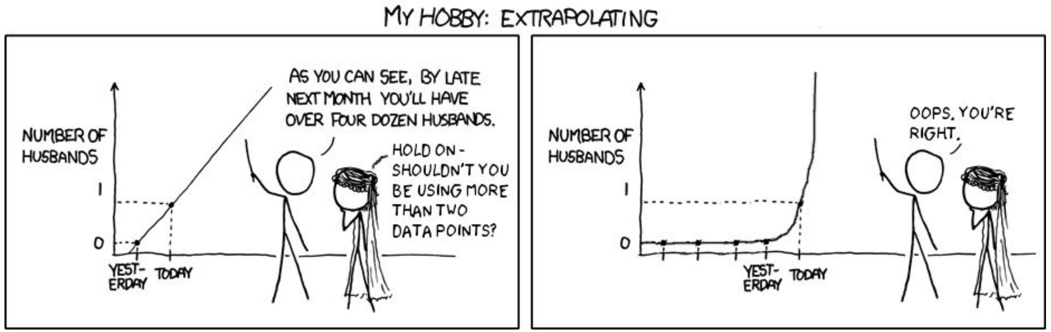
Solution: Client calculates the "present" state of the game based on the last state sent by the server, plus the inputs the server hasn't processed yet. Client discards requests the Server has acknowledged receiving.



How to Render Other Clients

Extrapolation/Dead Reckoning

- Treat others as physics objects and render using last known forces
- OK for deterministic games (racing)
- Bad for non-deterministic, high jerk games

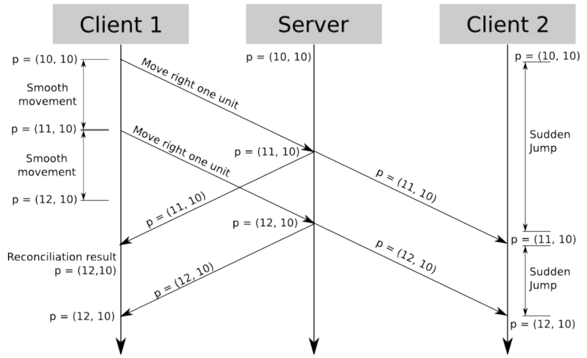

MY HOBBY: EXTRAPOLATING
NUMBER OF HUSBANDS
1
0
YEST-ERDAY
TODAY
AS YOU CAN SEE, BY LATE NEXT MONTH YOU'LL HAVE OVER FOUR DOZEN HUSBANDS.
HOLD ON - SHOULDN'T YOU BE USING MORE THAN TWO DATA POINTS?
NUMBER OF HUSBANDS
1
0
YEST-ERDAY
TODAY
OOPS. YOU'RE RIGHT.

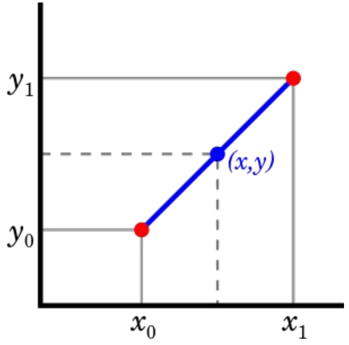
How to Render Other Clients

Interpolation: Render players based on old/previous authoritative data

Pros: Shows player movement more accurately

Drawbacks: Bouncing ball, dropped packets, other players rendered in past!





Lag Compensation

Player sees himself in the present, sees other players in the past (a bit)

Before executing any player command, the server:

- Computes player latency & moves him back in time
- Computes all other player lag (latency + interpolation) and moves them back in time relative to YOU
- Execute command and move everyone forward in time again

Lag Compensation

Design Tradeoffs

- Previously, had to lead enemies by an amount related to latency
- Now, the enemy can be killed when he thinks he is safe
    - HIghly lagged player shoots less lagged player and hits, after the LL has hidden behind corner
    - Usually not noticed: This is a “rare” occurrence, and players may not know where enemies aim

BaaS: Backend as a Service

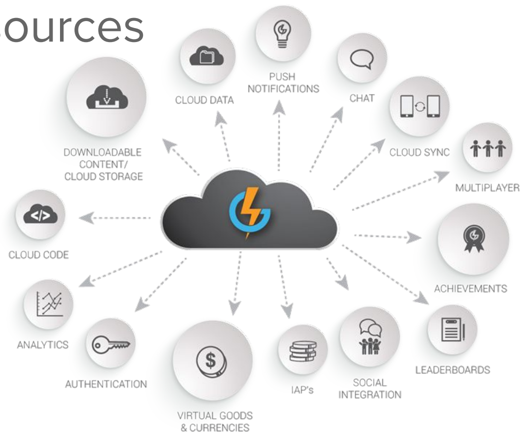
Out-of-the-box connections for games (and apps) to cloud services

Benefits

- Fast to prototype, cheap, scalable, multi-platform

Drawbacks

- BaaS providers may be constrained with location/resources
- Baas provider may go out of business
- Can be more expensive as your app grows



BaaS

unity

Firebase

Parse

kinvey

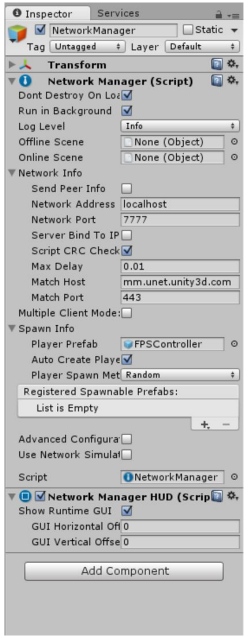
GAMESPARKS

GAMEDONIA

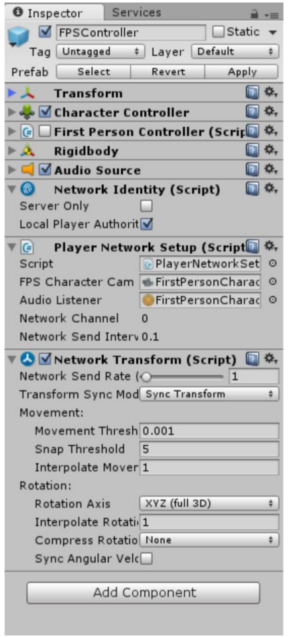
Kii

kumulos

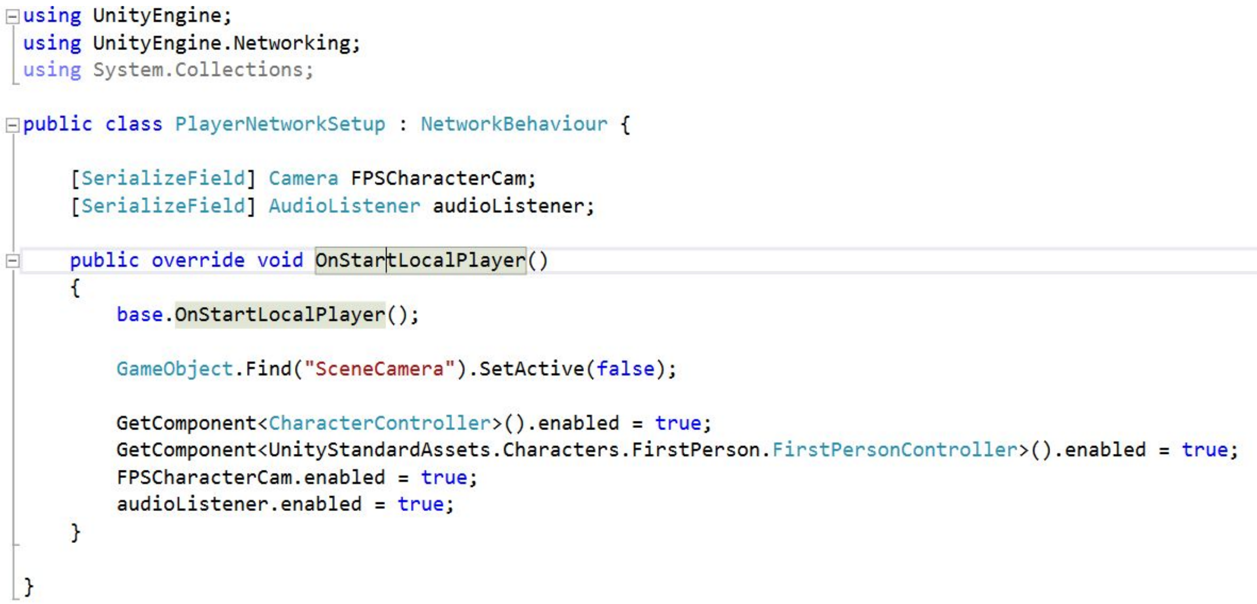
Unity Demo



```
Inspector    Services
[✓] NetworkManager            [ ] Static
Tag  Untagged    Layer  Default
Transform
Network Manager (Script)
  Dont Destroy On Loa   [✓]
  Run in Background     [✓]
  Log Level             Info
  Offline Scene         None (Object)
  Online Scene          None (Object)
  Network Info
    Send Peer Info      [ ]
    Network Address     localhost
    Network Port        7777
    Server Bind To IP   [ ]
    Script CRC Check    [✓]
    Max Delay           0.01
    Match Host          mm.unet.unity3d.com
    Match Port          443
  Multiple Client Mode: [ ]
  Spawn Info
    Player Prefab       FPSController
    Auto Create Playe   [✓]
    Player Spawn Met    Random
  Registered Spawnable Prefabs:
    List is Empty
  Advanced Configura    [ ]
  Use Network Simulat   [ ]
  Script                NetworkManager
[✓] Network Manager HUD (Scrip
  Show Runtime GUI      [✓]
    GUI Horizontal Off  0
    GUI Vertical Offse  0
Add Component
```



```
Inspector    Services
[✓] FPSController             [ ] Static
Tag  Untagged    Layer  Default
Prefab  Select  Revert  Apply
Transform
[✓] Character Controller
[ ] First Person Controller (Scrip
Rigidbody
[✓] Audio Source
Network Identity (Script)
  Server Only           [ ]
  Local Player Authorit [✓]
Player Network Setup (Script
  Script                PlayerNetworkSet
  FPS Character Cam     FirstPersonCharac
  Audio Listener        FirstPersonCharac
  Network Channel       0
  Network Send Interv   0.1
[✓] Network Transform (Script)
  Network Send Rate (   1
  Transform Sync Mod    Sync Transform
  Movement:
    Movement Thresh     0.001
    Snap Threshold      5
    Interpolate Mover   1
  Rotation:
    Rotation Axis       XYZ (full 3D)
    Interpolate Rotati  1
    Compress Rotatio    None
    Sync Angular Velc   [ ]
Add Component
```



```csharp
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;

public class PlayerNetworkSetup : NetworkBehaviour {

    [SerializeField] Camera FPSCharacterCam;
    [SerializeField] AudioListener audioListener;

    public override void OnStartLocalPlayer()
    {
        base.OnStartLocalPlayer();

        GameObject.Find("SceneCamera").SetActive(false);

        GetComponent<CharacterController>().enabled = true;
        GetComponent<UnityStandardAssets.Characters.FirstPerson.FirstPersonController>().enabled = true;
        FPSCharacterCam.enabled = true;
        audioListener.enabled = true;
    }

}
```

Server: port=7777
Client: address=localhost port=7777
Stop (X)

networking_in_action

with Quake: QuakeWorld

QuakeWorld

Overview

QuakeWorld
Overview

- QuakeWorld is a multiplayer update written by John Carmack for id software's game Quake.
- Quake was released on June 22, 1996 on MS-DOS. (yes DOS)
- QuakeWorld soon followed Quake's release being made available in December of 1996
- Quake had a very good playing multiplayer for people with broadband connections (very few people at the time) and LAN multiplayer games.
- The issue was that users with dial-up modem connections were experiencing issues due to latency problems. This is where the quake world update came in!
- The QuakeWorld update is considered to be the first popular online multiplayer FPS game.
- Was the first online multiplayer game I ever played.

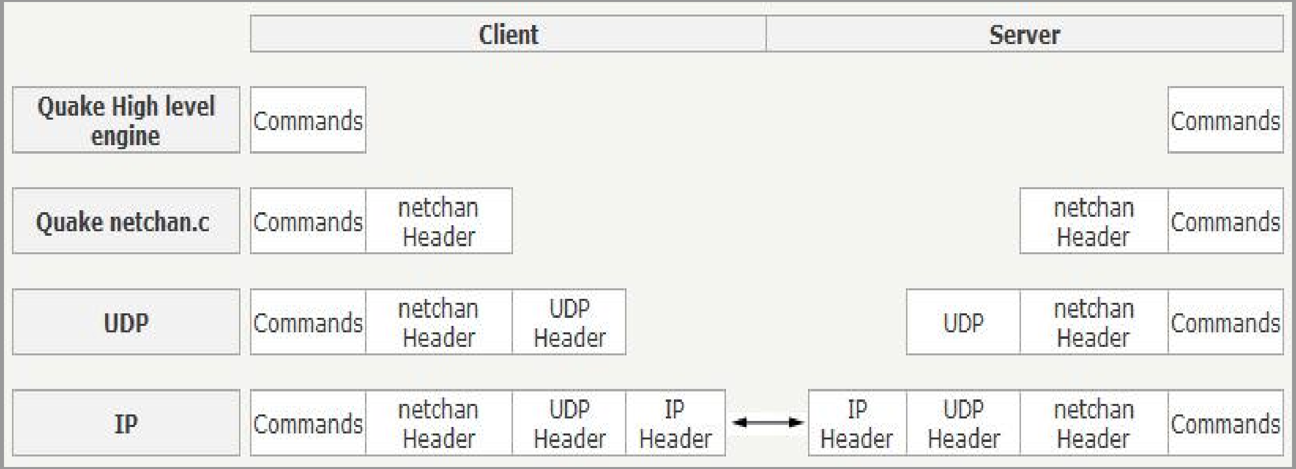
Quake
Overview

QuakeWorld
Networking : Introduction

- The QuakeWorld update to Quake is considered a “game” changer in the video game industry relative to networking.
- All future games used the same approach to networking following the release and success of QuakeWorld.
- In the original Quake multiplayer the strict TCP/IP was used as the medium which multiplayer games were played.
- Problems with TCP/IP…
- UDP Incoming!
- Success!

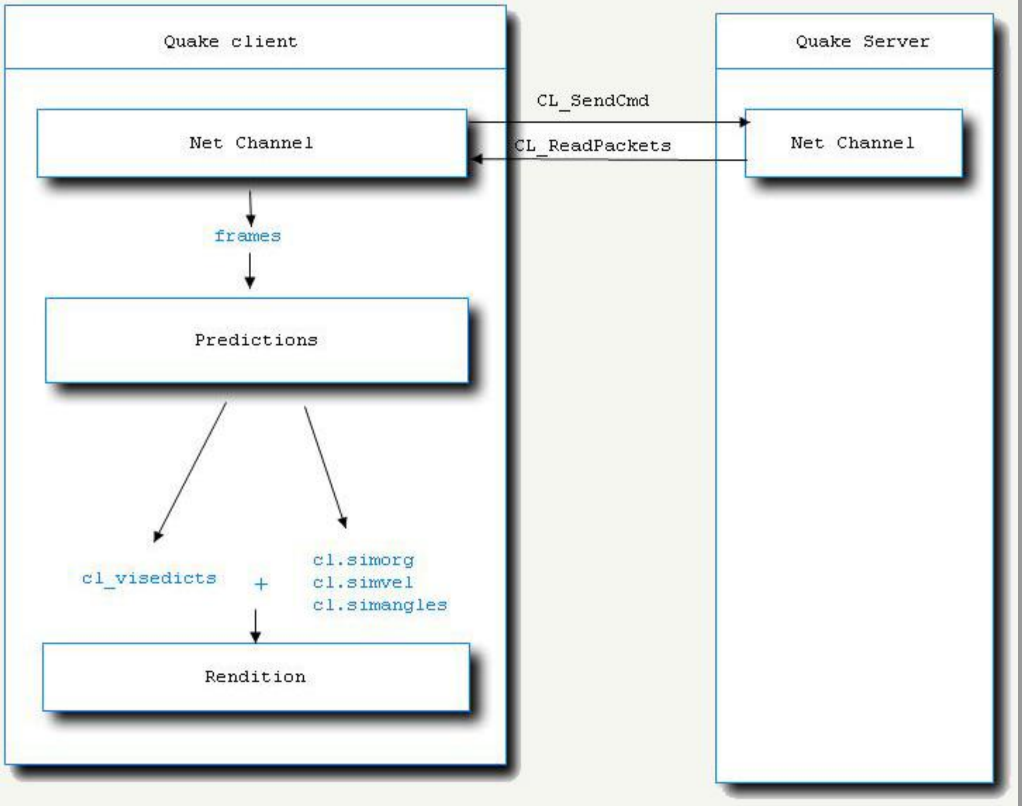
QuakeWorld
Networking : OSI Model

QuakeWorld’s Revamped OSI Model



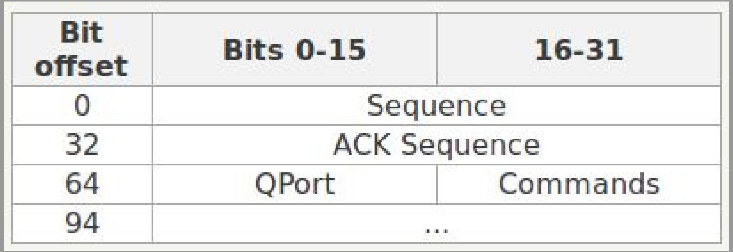
	Client					Server			
Quake High level engine	Commands								Commands
Quake netchan.c	Commands	netchan Header						netchan Header	Commands
UDP	Commands	netchan Header	UDP Header				UDP	netchan Header	Commands
IP	Commands	netchan Header	UDP Header	IP Header	⟷	IP Header	UDP Header	netchan Header	Commands

QuakeWorld
Networking : High-Level

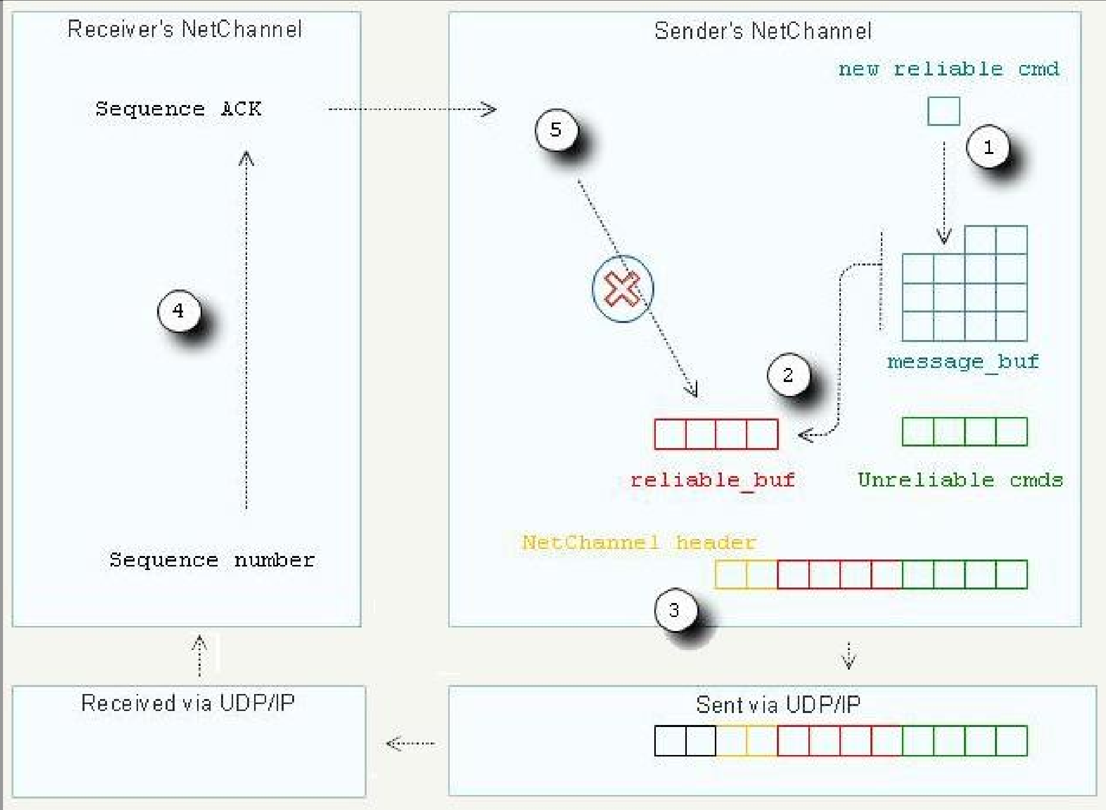


QuakeWorld
Networking : Netchan Layer

net_chan.c Header



Bit offset	Bits 0-15	16-31
0	Sequence	
32	ACK Sequence	
64	QPort	Commands
94	...	



QuakeWorld

Networking : Latency Calculation

net_chan.c

QuakeWorld
Networking : UDP Layer & Qport

One of the issues tackled by John Carmack as routers became more and more popular was use a remote IP in combination with a UDP port. He eliminated this issue. By replacing the UDP port with a Qport in the Net Channel layer’s header.

Located in **net_chan.c** source file once again.

QuakeWorld
Client

net_chan.c

QuakeWorld
Server

sv_main.c

sv_nchan.c

QuakeWorld

Linux Demo

QuakeWorld Gameplay

THE END