CMSC 498M: Chapter 9
Networking and Multiplayer Games

Reading:
- Network and Multiplayer, by Chuck Walters (Chapt 5.6 in "Introduction to Game Development" by S. Rabin)

Overview:
- Multiplayer games
- Networking: Protocols and packets
- Socket programming and Racknet
- Cheat Detection

Multiplayer Games

Multiplayer Games: A number of players communicating through a network

Persistent Games: Such as "World of Warcraft", where state is maintained, regardless whether anyone is playing

Transient Games: Only exist while people are playing, and reset each time the server-side is reset

Performance Issues:
- Latency: How much time delay until global state to be updated?
- Reliability: How often is data lost or corrupted?
- Bandwidth: What is the rate of data transfer?
- Security: How is the game-play protected from tampering/cheating?

Tradeoffs:
- All of these considerations interact, and trade-offs must be made
Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games

Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
    - Packets and Protocol Basics
    - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games
Multiplayer Factors: Event Timing

Factors in Multiplayer Games:
- **Event Timing**: How do player's interact with the game?
- **Shared Display**: How do different players perceive the game state?
- **Connectivity**: How are players connected and how do they communicate/interact?

**Event Timing**:
- **Turn-based**: Player’s move in turns (round robin). Other players wait.
- **Real-time**: Players move simultaneously.
  - May need to handle race-conditions (e.g., two players attempt to acquire the same resource at the same time).
  - “Twitch Games” typically require very low latencies, less than 150 ms.

Multiplayer Factors: Shared Display

Shared Display: for multiplayer games on a single platform

**Full Screen**:
- **Complete Player Visibility**: Such as board games and sports games. Everyone sees everything at all times.
- **Player Funneling**: Restricts players to a small region of game space. As players move this region moves with them.
- **Turn-Based Screen Control**: Active player controls the viewpoint.

**Split Screen**: Each player has a separate portion of display through the use of separate viewports.

**Components**: System maintains the following information for each player:
- **Camera**: Viewpoint
- **Cull Data**: What portion of the environment is visible
- **Heads-up Display**: Game stats displayed on top of scene
- **Map Data**: Centered about the current player

- **Audio Effects**: Need to be split among players as well.

**Issues**: Maintaining consistency between views. Practical only for a small number of players (e.g., 2-4).
Multiplayer Factors: Connectivity

**Connectivity**: How do players connect?

**Direct Link**:
- **What**: Connected directly typically through short connections
- **Examples**: Serial and USB cable, wireless (infrared and Bluetooth)
- **Pro**: Fast, reliable. **Con**: Few players, only small distances

**Circuit-Switched Network**:
- **What**: Unshared direct connection between endpoints
- **Example**: Traditional public telephone system
- **Pro**: Reliable, low-latency. **Con**: Low bandwidth, few players

**Packet-Switched Network**:
- **What**: Communication broken into small packets that are routed over a shared network
- **Example**: Internet
- **Pro**: Many players, large areas. **Con**: Variations in latency/bandwidth

Networking Basics

**Network**:
- A group of two or more computers connected together
- Henceforth we consider packet-switched networks

**Characteristics**:
- **Scale**: The area spanned by the network
  - **LAN** (Local-area network) E.g., connecting one business or school
  - **WAN** (Wide-area network) Connecting computers distributed over an entire city, state, country or the world
- **Topology**: How the computers are connected together. Examples: Ring, star, bus, tree
- **Protocol**: Agreed upon rules for communicating and routing information. Examples: TCP, IP, UDP, FTP, HTTP
- **Architecture**: General communication structure. Examples: Peer-to-peer, client server
Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games

Protocol

Protocol:
- A convention for routing and transferring data over packet-switched networks
- Communication networks may be unreliable and may connect machines having widely varying manufacturers, operating systems, speed, data formats

Issues:
- Packet sizes: Fixed or variable sized packets?
- Handshaking: Communication exchange to ascertain how data will be transmitted (format, speed, etc.)
- Acknowledgements: For receipt of data
- Error checking/correction: Handling errors in data transmission
- Compression: Reducing data size due to limited bandwidth
- Encryption: To protect private data
Packets

Packet:
- The logical transmission unit of a protocol
- Two parts: Header (information) and payload (data)
- Example: Quake network protocol packet structure:
  http://www.gamers.org/dEngine/quake/QDP/qnp.html

Simple Example:
```c
struct Packet {
    // Header
    short packetLength; // length of the packet (in bytes)
    short packetType; // E.g. data, control, acknowledgement
    int checksum; // checksum used for error checking
    // Payload
    char data[256]; // ...the data
};
```

Issues:

Serialization:
- Pointers and references cannot be reliably transmitted since they refer to local memory. Convert them to names or indices to an array
- Abstract data types: Are often based on references for inheritance

Endianness: Transmitting multi-byte numbers: 0123
- Little Endian: Low-order bytes first: 3, 2, 1, 0
- Big Endian: High-order bytes first: 0, 1, 2, 3

Irrelevant Trivia Alert: These terms come from the novel Gulliver’s Travels by Jonathan Swift on a civil war between Lilliputian factions that cracked their hard-boiled eggs from the big end or the little end.

Intrinsic Types: Use __int32 rather than int to force compiler to use 32-bit integers
- Unicode: Better than ASCII for string data
Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games

The OSI Model

**Open System Interconnect (OSI) Model**: Formalizes the multilayered structure of networks

**Application**: End-user processes (e.g., mail (smtp), ftp, telnet)
**Presentation**: Packetization, byte order, encryption, compression
**Session**: Connection and data exchange (e.g., logging in/out, socket)
**Transport**: Flow control (e.g., TCP/UDP)
**Network**: Basic routing (e.g., IP)
**Data Link**: Packet/frame structure
**Physical**: Physical medium (wire)
Physical Layer: Latency and Bandwidth

Physical Layer:
- The medium over which data is carried
- Examples: twisted-pair wire, coaxial cable, wireless

Latency and Bandwidth:
- Time of flight: Time to send a single bit of data.
  Includes delays at switches from source to destination
- Bandwidth: Maximum transfer rate from source to destination in bits per second (bps). Includes header
- Transmission time: messageSize / bandwidth
- Transport latency: timeOfFlight + transTime
- Sender (receiver) overhead: Time to process packet for sending (receiving)
- Total Latency: overheadsend + timeOfFlight + transTime + overheadrec
- Effective Bandwidth: messageSize / totalLatency

Example of Computing Effective Bandwidth:

Raw bandwidth: 10Mbps (mega-bits per second)
Sender overhead: 250 μsec (0.00025 sec)
Receiver overhead: 300 μsec (0.00030 sec)
Message size: 1000 bytes (8000 bits)
Distance: 1000km (Transmission speed = 150,000km/s)

What is the effective bandwidth?

\[
\text{TransTime} = \frac{8000\text{b}}{10\text{Mb/s}} = 800\text{b} = 800\mu\text{s}
\]
\[
\text{TimeOfFlight} = \frac{1000\text{km}}{150,000\text{km/s}} = 6667\mu\text{s}
\]
\[
\text{TotalLatency} = 250 + 6667 + 800 + 300 = 8017\mu\text{s} = 0.008s
\]
\[
\text{EffBandwidth} = \frac{8000\text{b}}{0.008s} = 1\text{Mb/s}
\]

Note that this is only 1/10th of the raw bandwidth
Physical Layer: Latency and Bandwidth

Maximum bandwidths of common media connection types:

<table>
<thead>
<tr>
<th>Media Connection Type</th>
<th>Maximum Bandwidth (bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial cable</td>
<td>20K</td>
</tr>
<tr>
<td>USB 1&amp;2</td>
<td>12M, 480M</td>
</tr>
<tr>
<td>ISDN</td>
<td>128K</td>
</tr>
<tr>
<td>DSL</td>
<td>1.5M down, 896K up</td>
</tr>
<tr>
<td>Cable</td>
<td>3M down, 256K up</td>
</tr>
<tr>
<td>LAN 10/100/1G BaseT</td>
<td>10M, 100M, 1G</td>
</tr>
<tr>
<td>Wireless 802.11 a/b/g</td>
<td>54M, 11M, 54M</td>
</tr>
<tr>
<td>Power line</td>
<td>14M</td>
</tr>
<tr>
<td>T1</td>
<td>1.5M</td>
</tr>
</tbody>
</table>

Note: Rates vary with direction

Typical Internet Latencies: 50-100ms

Actual delivery is around 70% of maximum.

Data Link Layer

Data Link Layer:
- Puts data in frames and ensures error-free transmission.
- Controls the timing of the network transmission. Adds frame type, address, and error control information.
- Examples of data link protocols are Ethernet for local area networks and PPP, HDLC and ADCCP for point-to-point connections.
- Network Interface Card (NIC): Performs these operations. Each NIC is associated with a MAC (media access control) address.
- All devices within a given subnetwork must have unique MAC addresses.

Source: Chapt 5.6 of Rabin

Chapter 9, Slide 17
Copyright © David Mount and Amitabh Varshney
Network Layer

Network Layer:
- Performs end-to-end (source to dest) packet delivery, (whereas the data link layer is for node-to-node)
- Performs network routing, flow control, data segmentation/de-segmentation, and error control
- Famous example: The internet protocol (IP)

IP Addresses:

IP version 4: (IPv4)
  - Address is 4 bytes, typically displayed in decimal: 255.8.128.16
  - Only $2^{32} = 4.3 \text{ billion}$ possible addresses

IP version 6: (IPv6)
  - Address is 16 bytes, displayed as 8 16-bit segments in hex: [2001:0db8:85a3:08d3:1319:8a2e:0370:7344]
  - Supports $2^{128} = 3.4 \times 10^{38}$ addresses. Roughly 1 address for every atom in everyone’s body on the planet

Chapter 9, Slide 19
Copyright © David Mount and Amitabh Varshney

Network Layer: Special IP Addresses

Unicast Address: An individual’s IP address. Sources:
  Static: Fixed address reserved for servers that require a well-known address. (Examples: DNS server, gateway router)
  Dynamic: Assigned dynamically by a DHCP server (dynamic host configuration protocol) to a specific MAC address (Typical case)

Special Addresses: Some IP addresses have special meaning:
  Multicast (224-230): Pensent to this address are routed to all members of a multicast group
  Local Broadcast 255.255.255.255: Packets sent to this address are routed to all members of the local network
  Loop-back 127.0.0.1: Packets sent to this address loop back to the current machine without entering the physical network (e.g., for testing)
  Domain Name: Human readable address (e.g., www.gamedev.net rather than 16.15.32.1). Stored in domain name server (DNS)

Chapter 9, Slide 20
Copyright © David Mount and Amitabh Varshney
Transport Layer

- Provides transparent transfer of data between hosts
- Responsible for end-to-end error recovery and flow control, and ensuring complete data transfer
- Makes the network layer’s services more reliable
- Provides notion of ports, as extension of IP addresses
- Examples: Transmission Control Protocol (TCP), User Datagram Protocol (UDP)

Ports:
- Ports are essentially ways to address multiple entities on the same node. Identified by an integer in the range 0..1023
- Net Address = IP Address + Port Number
- Each application ”listens” for information on a single port
- Can run multiple network applications at the same time
- Examples: ftp=21, telnet=23, smtp=25, http=80, https=433

Transport Layer: TCP

Transmission Control Protocol: (TCP) Features:
- Ordered-data transfer: Assign each packet a unique sequence number and reorder incoming packets in proper sequence order
- Error-free data transfer: Uses checksum to identify corrupted packets and sends negative acknowledgement
- Retransmission of lost packets: If receipt is not acknowledged
- Discarding duplicate packets: By detecting duplicate sequence nos.
- Congestion avoidance (flow control): When receipts are not received in a timely manner, the transmission rate is reduced. (Like having a traffic light controlling traffic onto a freeway)

Miracle of TCP/IP:
- These protocols were developed at a time when the Internet was much smaller, but they scaled up remarkably well to today’s internet
Transport Layer: TCP

TCP packet structure:

- **source port number**
- **destination port number**
- **sequence number**
- **acknowledgement number**
- **window size**
- **Internet checksum**
- **ptr to urgent data**
- **options**
- **data**

Transport Layer: UDP

**User Datagram Protocol (UDP):**
- Connectionless protocol with no guarantees of delivery
- "Send and forget" individual packets
- Faster than TCP: Smaller packets, lower overhead, lower latency
- Popular for games, since much state information is nonessential and quickly goes out of date

UDP packet structure:

- **source port number**
- **destination port number**
- **length**
- **checksum**
- **data**
Session Layer and Socket Programming

Session Layer:
- Manages connections between applications
- Responsible for establishing/terminating connections and coordinating data exchange

Sockets API:
- Library for low-level network programming and inter-process communication
- Berkeley Sockets API: released with 4.2 BSD release of the Unix operating system. Quickly became the de facto standard
- WinSock: Windows version of Berkeley sockets, with additional features for the Windows environment

Socket Programming

Things you can do with Sockets:
Create:
- You can specify whether the socket is TCP or UDP
Connect:
- Connect to a remote listening host by giving the net address (e.g., IP address and port)
- Support for converting domain names to IP addresses
Stream Transmission: (for TCP)
- Send and receive data through a socket
Datagram Transmission: (for UDP)
- Analogous operation for UDP sockets
What to send?
Game state/events: Depends on the specifics of the game
Presentation Layer

Presentation Layer:
- Responsible for delivery and formatting of information to application layer for further processing or display
- Relieves application layer of concern of syntactical differences in low-level data representations

Examples:
- **Format Conversion**: For example strings:
  Length+Text: "13,thisisastring"
  Null terminated: "thisisastring\0"
- **Packing**: E.g. packing small enumerations into bit fields
- **Float to fixed**: Convert floating point numbers to fixed point
- **Compress data structures**: E.g. encode computational forms (rotation matrices) to more concise representations (quaternions)
- **Encryption**: Protecting private data like passwords
- **Serialization**: Removing pointers and references

Resources

Further information on WinSock:
- "WinSock2 for Games": Tutorial on WinSock from gamedev.net:
  http://www.gamedev.net/reference/articles/article1059.asp
- "Game Programming with Asynchronous Sockets": Another one.
  http://www.gamedev.net/reference/programming/features/asyncsock/

Berkeley Sockets:
- Tutorial from RPI:
  http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html

General Network Programming:
- "The Internet Sucks:
  Or, What I Learned Coding X-Wing vs. TIE Fighter."
  http://www.gamasutra.com/features/19990903/lincroft_01.htm
Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games
Games and Networking

Industry Information:
- The top 17 game companies brought in $33B in revenue in 2005
- Major Players: Combined share $23B
  - Nintendo
  - EA
  - Sony (games part)
  - Microsoft (games part)

Online Games: About 50% of this is from online games:
- Massively Multiplayer Online Games (MMOGs)
- Smaller online play systems: Squad-on-squad play (under 32 players), or player-on-player games
- Massively Multiplayer Online Role-Playing Games (MMORPGs) about $4.3B

MMOG Subscription Growth

Source: http://mmodata.blogspot.com
**MMOG Subscription Growth**

![Graph showing MMOG subscription growth from 1998 to 2011 with peaks between 1,000,000 and 12,000,000 subscribers.](http://mmodata.blogspot.com)

*Source: http://mmodata.blogspot.com*

---

**MMOG Rationale**

**Why are MMOGs Popular?**

- Humans are better at strategy than AI engines are:
  - Less predictable for general settings

**Social Interaction:**

- Peer group support
- Natural language communication
- Development of game “culture”

**Make money:**

- Professional gamers
- Virtual economies
- …and the inevitable rise in “virtual crime”
Origins of MMOGs

Quake (1996):
- **First** widely used 3D multiplayer online game
- **Difficult to find game servers:**
  - Gamers exchanged IP addresses by email or gaming websites
- **No persistent state:**
  - Short-lived ad-hoc fight-to-death sessions

Advent of MMORPGs:
- The Realm Online, Meridian 59, Ultima Online, Underlight and EverQuest in the late 1990s

Scale:
- 1995: 500 simultaneous players
- 2000: Thousands
- 2007: Hundreds of thousands

Persistent MMOGs

Persistence:
- Players can **join and leave** as they choose
- Each player can **affect the persistent world** and be affected by it
- Plot/events **progress** even while a player is offline
- Concept started in MMOGs and later began to be used even in offline games (use the game clock)
- Requires commercial 24/7 **game servers**
- Led to **virtual currency**, **botting**, ...
Challenges of MMOGs

Need to support millions of subscribers:
...a few 100K concurrently

Back-End Networking:
- Authentication and billing
- Ranking / black lists
- Run-time mods/patches
- Guard against denial of service attacks

Front-End (in-game) Networking:
- Network topology (client-server, peer-peer)
- Persistence
- Latency, bandwidth
- Virtual economy (audits, gold farming, ...)
- Distributed protocols

Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games
Peer-to-Peer

Peer-to-peer Architecture:
- Each player communicates directly with all other players

Possible complexity:
- $O(n^2)$
- Limited scalability

Advantages:
- Low latency, robust

Disadvantages:
- Demands high bandwidth, limited scalability
- Persistence vs. redundancy tradeoff

Client-Server

Client-Server Architecture:
- Each player (client) communicates with server

Advantages:
- Scalable, usually requires less bandwidth
- Easier to provide persistence

Disadvantages:
- Server bottleneck: Higher latency
- Server failure: Lower reliability

Client Types:
- Thin Client:
  - All simulation on server
  - Good for resource-poor clients: Cell phones, PDAs
- Simulation Client:
  - Most simulation and world distributed amongst clients
  - Server maintains/update state

Figure source: Mike Zyda

Chapter 9, Slide 39
Copyright © David Mount and Amitabh Varshney
**Multiple Servers**

**Multiple Server Architecture:**
- Many distributed servers, each supporting a subset of the clients

**Advantages:**
- Reduced latency
- Scalability to millions of players
- Improved robustness

**Disadvantages:**
- Difficult to maintain consistency for persistence

**Possible Architectures:**
- Shards
- Mirrors
- Grids

---

**Multi-Server Shards**

**Multi-Server Shards:**
- Each server simulates a different instantiation of the world
- Each player plays within his own realm, and does not influence the other worlds
- Minimal server-server traffic.

**Origin:**
- "...different images of the world, trapped in the shattered shards of a mystic gem..." Ultima Online story

**Examples:**
- Ultima Online
- Neverwinter Nights
- Silkroad Online
- World of Warcraft (called Realms)
Multi-Server Mirrors

Multi-Server Mirrors:
- Each server mirrors/replicates the same persistent world

Advantages:
- Common world view regardless of hosted server

Disadvantages:
- Heavy server-server traffic required for synchronization

Examples:
- Mankind
- PlanetSide
- A Tale in the Desert
- Entropia Universe

Multi-Server Grid

Multi-Server Grid:
- Each server (Sim) hosts a different region of the same persistent world
- Sims communicate with each other on a grid structure
- Users are transferred between servers as they move around

Advantages:
- Scalability for very large worlds

Disadvantages:
- Load balancing for high density regions

Examples:
- Second Life
Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games

Distributed Virtual Worlds

Distributed Virtual Worlds:
- Implementation is distributed, but appearance should be local
- Need to hide artifacts that suggest otherwise
- Each player sees real-time activity and reacts
- Actions should have immediate impact on the virtual world

Major Challenge:
- Latency: Delays induced by network structure
Distributed Virtual Worlds: Latency

Latency:
- All information received is \textit{out-of-date}
- Delays are variable and \textit{unpredictable}
- Maintaining the illusion of a consistent and persistent world is \textit{much harder when players interact}

Collisions:
- \textbf{Simple}: Bullets, missiles (linear or parabolic)
- \textbf{Complex}: Sword fighting, handshakes
- \textbf{Nightmare}: Successive/concurrent collisions/responses

Visual latency:
- 30Hz or higher (~ 30 ms)

Response times: Depend on interaction speed
- \textbf{Real-Time Strategy}: < 250 ms preferred, 250-500 ms playable, > 500 ms noticeable
- \textbf{First-Person Shooter}: < 150 ms preferred
- \textbf{Car race games}: < 100 ms preferred; 100 - 200 ms sluggish; > 500 ms out of control

Predictability:
- \textbf{Predictable} (even if sluggish) response considered better than variable (even if sometimes very fast)

Haptic (touch) latency:
- \textbf{Preferred}: Update rates > 1 KHz (< 1 ms) in force-feedback systems
Distributed Virtual Worlds: Challenges

Collision detection and response:
- Did the collision occur?
- If it did, when?
- Velocities, forces, in play at that instant

Environmental effects in a changing world:
- Global illumination
- Aural (sound) rendering
- Both can be computationally intensive

Managing Shared States: Three methods...

Shared repositories:
- All servers maintain a common consistent description of the world
- Absolute consistency!

Blind Broadcasts:
- Owner of each state transmits its current state at regular intervals
- Clients cache the most recent update information
- No acknowledgements
- Frequent state updates compensate for lost packets (hopefully)

Dead Reckoning:
- Transmit state updates occasionally
- Extrapolate from past updates to estimate the true shared state
- Need to correct state when an update is received
Dead Reckoning

Classical:
- Navigation with just logs, compass, clock (local/dead information) without looking at the sky (global/live information)

Distributed Worlds:
- Each processor predicts the state of the other prior info
- Corrects course when live information arrives
- Predict → Correct → Converge

Distributed Virtual Worlds: Challenges

Mutually Conflicting Goals:
- We can have either a dynamic world or a consistent world, but not both

Design Implications:
- For a highly dynamic shared state, hosts must transmit more frequent data updates
- To guarantee consistent views of the shared state, hosts must employ reliable data delivery
- Available network bandwidth must be split between these two constraints
Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- **Socket Programming and Cheating**
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games
Sockets Programming

Sockets:
- Provide a means for programs (running perhaps on different machines) to communicate with one another

Socket types:
Stream sockets: (TCP: Transmission Control Protocol)
  - Reliable two-way connected communication streams
  - Items arrive in the order they were sent
  - Virtually error-free

Datagram sockets: (UDP: User Datagram Protocol)
  - No connection - Just generate a packet and send it
  - Packets may not arrive in the order they were sent
  - Packets may not arrive at all

User Datagram Protocol (UDP)

Most real-time games use the UDP protocol:
- Faster and with lower overhead than TCP. Great!

But there are consequences: UDP packets...
  ... are not guaranteed to arrive
  ... are not guaranteed to arrive in order
  ... are guaranteed to arrive with correct data, but have no protection from hackers intercepting and changing the data once it has arrived
  ... do not require a connection to be accepted. (Assists cheating. For example, intercept the packet "Give ...blah... invulnerability," generate a copy of the message, and send it to the server anytime)

...and global consequences:
- Unlike TCP, UDP does not provide flow control or aggregation, and so it is possible to overrun the recipient’s capacity
Socket APIs

Low-level Socket APIs:
- Berkeley Sockets API: for Unix
- WinSock: for Windows
  - Both provide essentially the same functionality

Basic capabilities: (for Berkeley sockets, but Winsock similar)
- `socket(...)`: Create a socket (of either type)
- `gethostbyname(...)`: Map hostname (e.g., "mysite.com") to IP address
- `bind(...)`: Connect a socket to a port on your local machine
- `connect(...)`: Connect to a remote machine and port
- `listen(...)`: Wait for input to arrive from a socket
- `accept(...)`: Accept a request from another host to connect to you
- `send(...) / recv(...)`: Send and receive data
- `sendTo(...) / recvFrom(...)`: Send/receive (for datagram sockets)
- `close(...)`: Terminate communication

Sockets Program Structure

Program Structure:
- Typical client-server program structure (assuming TCP connection)

UDP is even simpler: Since
  - `listen`, `accept`, `connect` are not needed. Replace calls to `send/recv` with calls to `sendTo/recvFrom`
Overview

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
    - RakNet
      - Cheating in Multiplayer Games

RakNet: More Reliable UDP

RakNet:
- C++-based, open-source toolkit for (higher-level) UDP-based socket programming
- Supports client, server, and peer-to-peer communication
- Provides a layer over UDP, which addresses many of UDP’s shortcomings

RakNet Enhancements:
- Can automatically resend lost packets
- Can automatically order packets that arrived out of order
- Protects transmitted data, and inform the programmer if that data was externally changed
- Provides a connection layer that blocks unauthorized transmission
- Transparently handles network issues such as flow control and aggregation (grouping many small transmissions into one packet)
RakNet: Quick Overview

RakNet Standard Headers:

```c
#include "MessageIdentifiers.h"
#include "RakNetworkFactory.h"
#include "RakPeerInterface.h"
#include "RakNetTypes.h"
```

RakPeerInterface:
- The main RakNet object
- You will usually only generate one of these

```c
RakPeerInterface* peer = RakNetworkFactory::GetRakPeerInterface();
```
- Base class for more specific objects: RakPeer, RakClient, and RakServer

RakNet: Client Connection

Connection as a Client:
- Start up the network threads

```c
peer->Startup(1, 30, &SocketDescriptor(), 1)
```
- 1: maximum number of connections. For a pure client, we use 1
- 30: thread sleep time (in msec)
  - 0 msec: good for games that need fast responses, such as FPS.
  - 30 msec: good response times with little CPU usage
- `SocketDescriptor()`: specifies the port/IP-address to listen on.
  Since we want a client, we don’t need to specify anything
- 1: Force RakNet to use a particular IP as host
RakNet: Client Connection

Connection as a Client: (cont.)
- Connect to the RakNet server
  peer->Connect( serverIP, serverPort, 0, 0 );
- serverIP: IP address of the server
  - Use "127.0.0.1" or "localhost" to connect to your own machine (testing)
- serverPort: Port you want to connect to on the server
  - Any unused port number (in the range 0 to 2^16-1). Many are used by existing applications (ftp, smtp, http, etc.)
  - Ports over 32000 are generally open to whoever wants them
  - E.g.: serverPort = 60005, clientPort = 60006
- Last two arguments used for passwords.
- Only initiates the asynchronous connection process. You will receive:
  - ID_CONNECTION_ACCEPTED if successful, and
  - ID_CONNECTION_ATTEMPT_FAILED if not

RakNet: Server Connection

Connection as a Server:
- Connect to the RakNet server
  peer->Startup( maxConnectionsAllowed, 30, &SocketDescriptor( serverPort, 0 ), 1 );
  peer->SetMaximumIncomingConnections( maxPlayersPerServer );
- maxConnectionsAllowed: Maximum simultaneous connections
- 30: thread sleep time (in msec)
- SocketDescriptor: Which port to listen to
- maxPlayersPerServer: Maximum incoming connections to allow
**RakNet: Read a Packet**

Read a Packet:

```c
Packet* packet = peer->Receive();
```

- Returns 0 (NULL) if no input
- Data may come from engine or other RakNet instances

**Packet Structure:**

```c
struct Packet {
    SystemIndex systemIndex;  // Server only: Sender index
    SystemAddress systemAddr; // Who sent the packet
    unsigned int length;      // Data length in bytes (Deprecated)
    unsigned int bitSize;     // Data length in bits (Use this)
    unsigned char* data;      // The data (Cast as needed)
    bool deleteData;          // Internal use
};
```

First byte indicates data type

---

**RakNet: Send a Packet**

Send a Packet:

```c
const char* message = "Hello World";
peer->Send((char*) message, strlen(message)+1,
            HIGH_PRIORITY, RELIABLE, 0,
            UNASSIGNED_SYSTEM_ADDRESS, true);
```

- **message**: data to be sent
- **strlen(message)+1**: length of data. Allow one additional byte for string's null ('\0') terminator
- **HIGH_PRIORITY**: Packet priority (also "LOW" and "MEDIUM")
- Reliability options:
  - **UNRELIABLE**: may not arrive and order of arrival arbitrary
  - **UNRELIABLE_SEQUENCED**: in order, but may not arrive
  - **RELIABLE**: guaranteed arrival, but order is not
  - **RELIABLE_ORDERED**: guaranteed arrival, and in order
  - **RELIABLE_SEQUENCED**: out of order packets are deleted
- Final arguments indicate a broadcast to all RakNet systems
RakNet: Shutting Down

**Shutting Down:**

```
peer->Shutdown( 300 );
...
RakNetworkFactory::DestroyRakPeerInterface( peer );
```

- **Shutdown:** closes the connection. The connection can be restarted, using `Startup()`
  - **300:** Indicates how long to wait (in msec) for remaining packets to be sent
- **DestroyRakPeerInterface:** Shut RakNet down and free all memory

For more information:
- See [Raknet Manual and Tutorials](#). (Warning: Doxygen documentation appears to be out of date.)

---

**Overview**

- Multiplayer Games and Networking
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- Networking and MMOGs
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- Socket Programming and Cheating
  - Socket Programming
  - RakNet
    - **Cheating in Multiplayer Games**
Why Care About Cheats?

Achieving financial advantage:
- Competitive games with prizes are the obvious example (casinos)
- Virtual Economies:
  - People play the game, build good characters, and then auction them on eBay
  - If they can cheat to obtain better characters, they are achieving unfair financial advantage

Ruining your game play ⇒ ruining your profits:
- Online gaming is big business
- Players tend to have a strong sense of fairness
- If they believe they are being cheated, they will stop playing, and you will not make any money

Cheating is principally an online issue:
- Single player cheaters only affect themselves, so who cares?

Observations About Cheating

Pritchard’s Rules (Gamasutra article):
1. If you build it, they will come—to hack and cheat
2. Hacking attempts increase as a game becomes more successful
3. Cheaters actively try to control knowledge of their cheats
4. Your game, along with everything on the cheater’s computer, is not secure — not memory, not files, not devices and networks
5. Obscurity ≠ security
6. Any communication over an open line is subject to interception, analysis and modification
7. There is no such thing as a harmless cheat
8. Trust in the server is everything in client-server games
9. Honest players would like the game to tip them off to cheaters, hackers hate it
Common Cheating Attacks

**Reflex Augmentation:**
- Improve physical performance, such as the firing rate or aiming

**Authoritative Clients:**
- Clients issue commands inconsistent with the game-play, or mimic the server

**Information Exposure:**
- Clients obtain/modify information that should be hidden

**Compromised servers:**
- A hacked server biases game-play towards the group that knows of the hacks

**Bugs and Design Loopholes:**
- Bugs and design flaws are exploited

**Environmental Weaknesses:**
- Differences or problems with the OS or network environment are exploited

---

Reflex Augmentation

**Reflex Augmentation:** (a.k.a. Aimbots)
- Turning yourself into "the Terminator"
- Examples:
  - Aiming proxies: intercept communications, locate players, and shoot
  - Rapid-fire proxies: take each shoot packet and replicate it

**Fix #1:**
- Server validates player actions. Disqualifies players "too good" to be human

**Fix #2:**
- Make it hard to insert invalid network packets
- Encrypt packets: Must be fast, and so may be easy to crack
- Encryption dependent on the game state or some random value
- Serialize packets with a unique sequence of numbers. Hacker cannot copy or insert extra packets. (Requires reliable protocol)
Encryption

Typical Encryption:
- A key, known only to intended users, is used to convert regular data into something that appears random
- Hard to use encrypted data to obtain key or the original data

How to come up with the key:
- Agree on it ahead of time, e.g. when software is purchased
- Transmit it — key-exchange algorithms
- Derive it from somewhere else in such a way that all parties derive the same key (e.g. from game state)

Most encryption algorithms work on blocks of a fixed size:
- Split large amounts of data into smaller blocks
- Pad blocks that are too small

Authoritative Clients

Authoritative Clients: (Example)
- One player’s game generates bogus definitive event: e.g. "Player 2 just got 10,000 hit points."

How to Hack the Client:
- Alter executable. Change game data in other files. Hack packets

Fix: Insert command request steps:
- Player requests an action, its validity is checked, it is sent out on the network, and added to the player's pending event queue
- Incoming actions also go on the pending queue
- Actions come off the pending queue, are validated again, and then are implemented

If validation is hard to get right, try synchronization:
- Occasionally send complete game state around, and compare it
- More practical: send something derived from complete game state
**Information Exposure**

**Accessing/Modifying Hidden Parameters:**
- Modify the renderer to make walls transparent, modify maps to remove the fog of war
- Display variables are stored/modified in memory, or read out and displayed elsewhere
- Hackers use debugging tools to find the locations of key data in memory, and modify them transparently

**Fixes:**
- Check that players agree on the values of certain variables, and the validity of actions (synchronization again)
- Check for invalid actions based on the correct display. (E.g., aiming through walls)
- Compute statistics on drawing, and check (e.g. # of polygons drawn)
- Encrypt data in memory to avoid passive attacks

---

**Environment “Tweaks”**

Decluttering the environment by disabling texture mapping.

Return to Castle Wolfenstein
Compromised Servers

Customizable Servers:
- **Fact:** Some servers have **customization options**, and the community is allowed/encouraged to modify the server
- **Fact:** This is completely **legal**

Naïve Users:
- Do not have the skills or knowledge to check whether the server they are playing on is **altered**
- Will grow **frustrated**, blame the developer, and complain to friends

Illegal Modifications:
- if (player.name->contains("My_Clan") Damage = Damage * 0.80;

**Solution:**
- Warn players as they connect to the server, of any **non-standard modifications** (discovered through validation)

Exploiting Bugs and Design Flaws

**Bugs:**
- Some bugs enable **cheating**, such as a bug that enables faster weapon reloading, or one that incorrectly validates commands

**Poor Design Decisions:**
- Embedding **cheat codes** in single-player mode makes it easy for a hacker to track down the variables that control cheats
- Poor networking or event handling can allow **repeat commands** or other exploitations
- Example (Age of Empires and Starcraft): All resource management is done **after** all events for a turn are processed. Poor networking allowed **multiple cancel events** on the queue, which restored multiple resources

**Solution:**
- **Avoid bugs** and think carefully about the **implications** of design decisions on hacking
Environmental Weaknesses

Environmental Weaknesses:
- Facilities to deal with the OS or network may leave you vulnerable to some forms of attack.
- Example: (Firestorm) Generate a message from system's clip-board containing non-printing characters. Sending it to another user causes his program to abort.
- Interaction with almost any scripting language may leave you open to hacks not related to the game. (Your game could be a way in).
- Network connection drops or overloading can cause problems.

Targeted/Indiscriminant Cheats:
- Some cheats destroy the game for all players, which can be useful if you are losing.
- Others affect a specific opponent. (E.g., your worst enemy).

The Moral of the Story

Final Thoughts:
- You cannot prevent cheating completely.
- Try to make cheating as hard as possible (e.g., as hard as writing a new game).
- Do not trust information from others.
- Limit the potential damage.
- Test for anomalous/unrealistic behavior.
Summary

- **Multiplayer Games and Networking**
  - Multiplayer Game Basics
  - Packets and Protocol Basics
  - OSI Network Structure
- **Networking and MMOGs**
  - Multiplayer Game Overview
  - Online Game Architectures
  - Distributed Virtual Worlds
- **Socket Programming and Cheating**
  - Socket Programming
  - RakNet
  - Cheating in Multiplayer Games