UNIVERSITÉ Concordia UNIVERSITY

COMP 476

Advanced Game Development

Session 7

Networking and Multiplayer Games

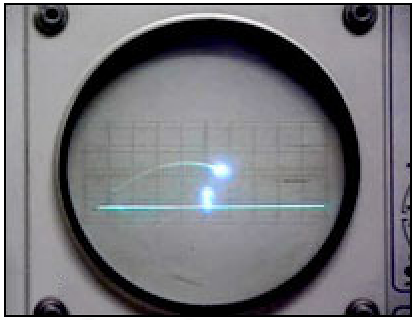
(Reading: See References)

Lecture Overview

- Networking
- Networking Computers
- Online Multiplayer Games
- Security
- Using Unity

Multiplayer Game?

- **More than one player at a time (or not) playing the same game session**
- **Tennis for Two is the *very first multiplayer game*, running on an oscilloscope (1958)**
- **Spacewar! is the *first multiplayer game running on a computer* (1961) (made >$100K)**



Tennis for Two

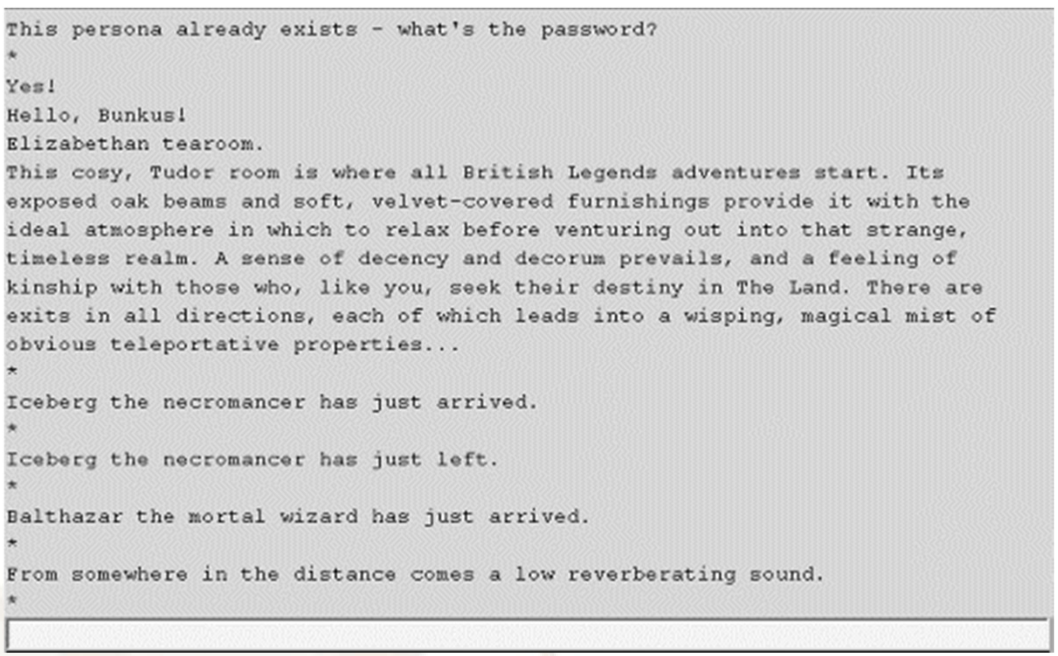
Taken from http://en.wikipedia.org/wiki/Tennis_for_Two

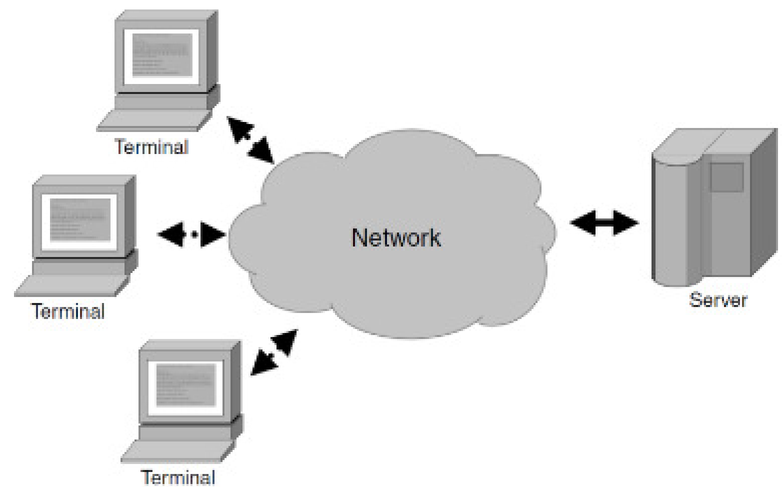
Spacewar!

Taken from https://history-computer.com/ModernComputer/Software/Spacewar.html

Online Game?

- Where players connect remotely to devices to play
- MUD-1: one of the first online games, text based
- Basic client-server topology (precursor to MMORPGs)





MUD-1 (1978)

Online Multiplayer
Milestones

- More than one player playing *on the same game session*
- First example: **Doom** by id software (1993)
  - Using IPX for communication
  - P2P (peer to peer topology)
  - Every *1/35th of a second*, game collects user input and broadcasts the network packets (sends it to other players)

  - **No server, no client** (or vice versa)

Online Multiplayer

Milestones

- **Quake (1996)**
  - **Big leap in online gaming**
  - ***Server-client topology***
  - **Used Internet protocols (TCP/IP) which enabled players to *play via the Internet***
  - **No need to meet in the same time and game room, Quake had its *own game rooms on the Internet***
  - **Clients send only their own inputs to the server and *get back the new game state from the server***

Online Multiplayer

Milestones

- **Ultima Online (1997)**
  - First popular Massive Multiplayer Online Role Playing Game (MMORPG)
  - Evolved out of MUDs
  - *Server-client topology*
  - Used Internet protocols which enabled players to play via the Internet
  - Game client uses an "isometric" perspective
  - Genre gained widespread popularity with EverQuest (1999), and then World of Warcraft (2004)

Networking

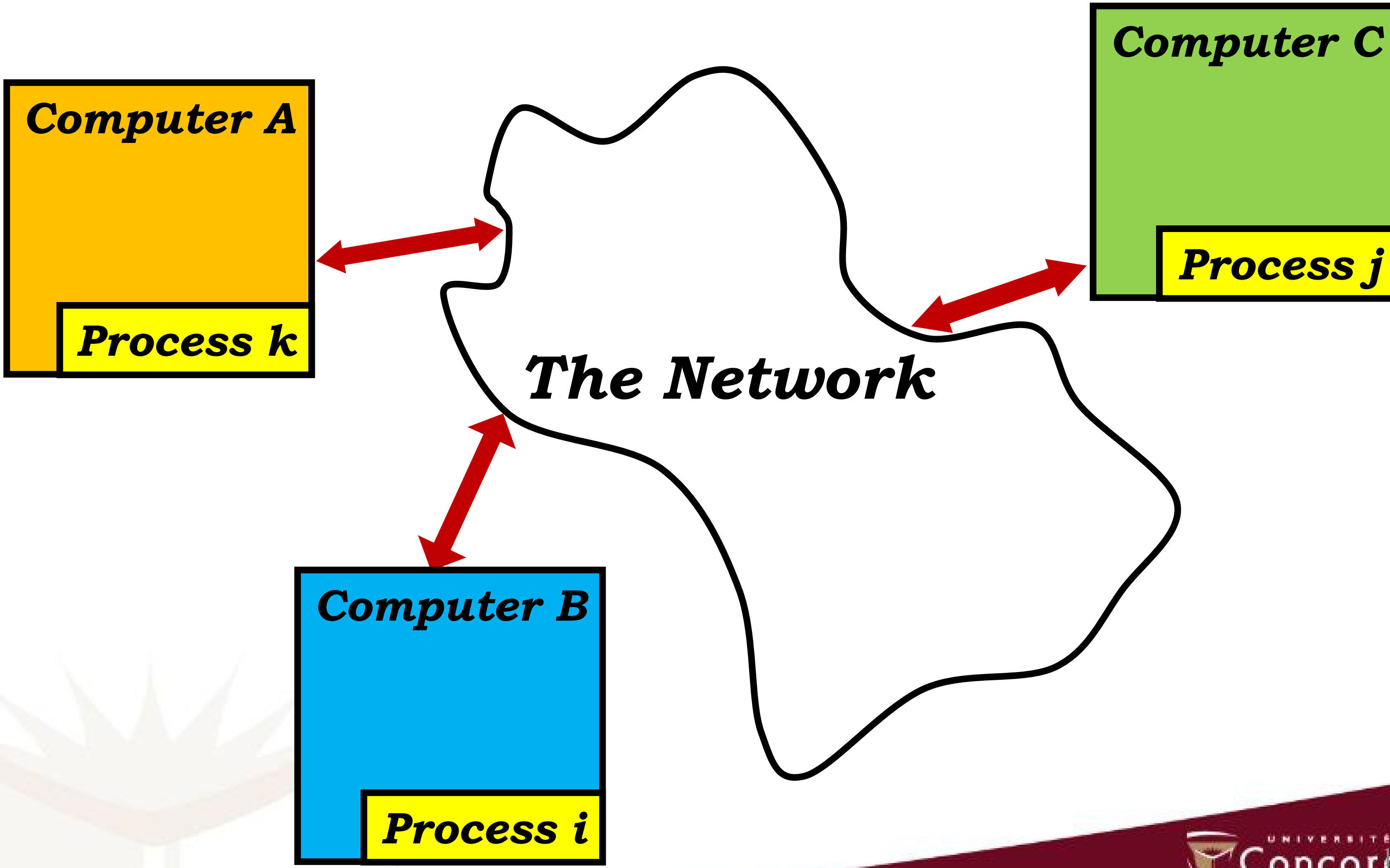
Context

- Networking was formerly regarded as "just another form of I/O"
- Distributed Computing
  - Shared files and other resources among physically separated systems ***on networks***
    - Network File System (NFS), remote printing, *etc.*
  - Integrated computations across network
    - Airline reservations, ATMs, *etc.*
  - Interactive games and multimedia
  - …
- Networked/online games may, or may not, be multiplayer games.

Network Goal

- Allow activities on multiple computer systems to communicate with each other
  - Shared memory, files, or data
  - Message (packet) passing
  - Remote Procedure Call (RPC)
  - Streaming
- Create abstractions that make these (relatively) transparent

Network Goal



Definition — Protocol

- These Networking Abstractions are defined *in terms of protocols*.
- **Protocol**: Formal set of rules that govern the formats, contents, and meanings of messages from computer to computer, process to process, *etc.*
- *Must be agreed upon by all the parties* to a communication
- May be defined in terms of other protocols

Protocol Design

- Packet Length Conveyance
- Acknowledgement Methodology (if applicable)
- Error Checking / Correcting (if applicable)
- Compression
- Encryption
- Packet Control (*e.g.*, order of the packets)

Plethora of Protocols

- TCP, UDP, IP, IPv6, NCP, SMTP, SNNP, NNTP, FTP, TFTP, POP, IMAP, HTTP, VMRL, …
- AppleTalk, Netware, …
- RPC, NFS, …
- CORBA, GLOBE, JINI, …
- Network Streaming, …
- …
- How to make sense out of all of them?

Network Stack

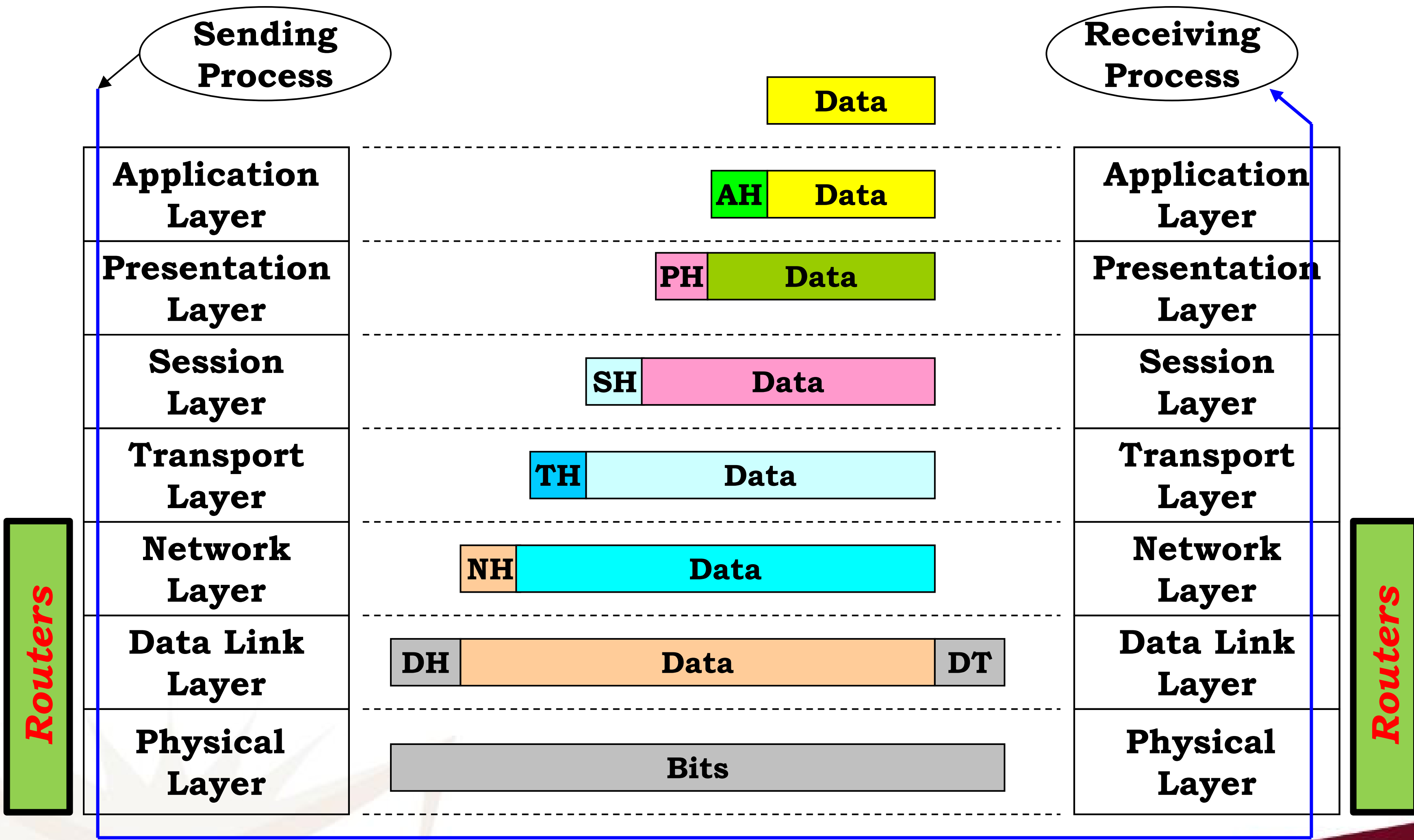
- 1983 – Open System Interconnection (OSI) seven layer Reference Model
  - Working group of the International Standards Organization (ISO)
  - Defines seven layers
    - Describe how applications communicate with each other
      - Via network-aware devices

OSI 7-Layer Model

- Primarily *__a software and protocol architecture__*
- **Layers** of model *correspond to layers of abstraction*
- Number of layers:–
  - Large enough
    - *__Distinct functions need not be thrown together__*
  - Small enough
    - *__Architecture does not become unwieldy__*

Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Example of OSI Model



Example of OSI Model

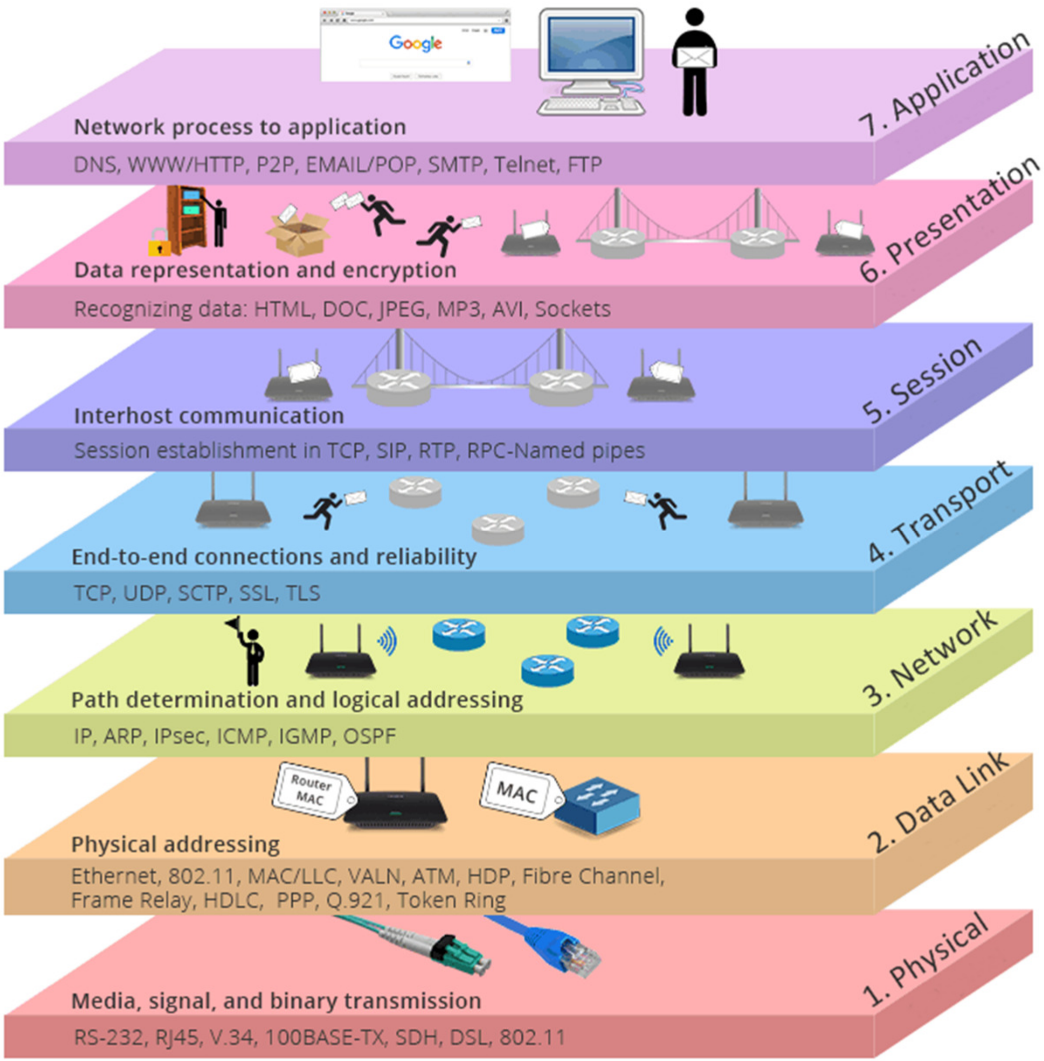


Image from https://community.fs.com/

Protocol Stack

- Application Layer
- Presentation Layer
- Session Layer
- Transport Layer
- Network Layer
- Data Link Layer
- Physical Layer

Physical Layer

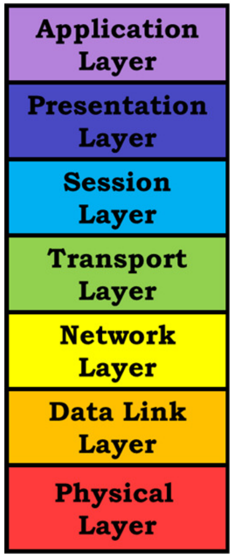
- **Bandwidth**
  - Width of data pipe
  - Measured in bps = bits per second
- **The Medium**

	Serial	USB 1&2	ISDN	DSL	Cable	LAN 10/100/1G BaseT	Wireless 802.11 a/b/g	Power Line	T1
Speed (bps)	20K	12M 480M	128k	1.5M down 896K up	3M down 256K up	10M 100M 1G	b=11M a,g=54M	14M	1.5 M

Table: Max Bandwidth Specifications

- **Latency**
  - Travel time from point A to B
  - Measured in Milliseconds

Protocol Stack

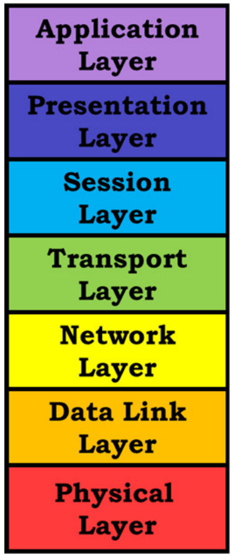


Network Layer

- Packet Routing
  - Multi-”hop” connections
  - Routers, Hubs, Switches
- Internet Protocol (IP)
  - Contains Source & Destination IP Address
  - IPv4 (*e.g.*, 194.153.205.26)
    - Widespread Infrastructure still
  - IPv6 (*e.g.*, 3ffe:1900:4545:3:200:f8ff:fe21:67cf)
    - Larger IP address

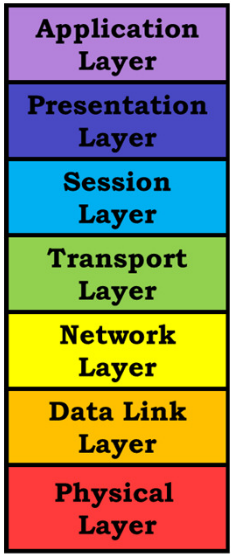
Protocol Stack

Session Layer (DNS)



- **Domain Name Service**
  - **Converts *device's* text *name* *to IP address***
  - **Local cache resolution possible**
  - **<u>Otherwise</u>, <u>must</u> contact one/more DNS servers to resolve**
- **<u>Game Tips</u>**
  - ***Store local game cache* to use when DNS is out of order.**
  - ***DNS resolution often slow*, use cache for same day resolution.**

Protocol Stack



Application Layer

- **Handles Game Logic**
- **Update Models for Game State**
  - **Input Reflection**
  - **State Reflection**
- **Synchronization in Games**
  - **Make sure everyone is playing *the same game session***
  - **Dead Reckoning, *etc.***
  - **We'll explore these in detail later (next class) when we consider networked multiplayer games…**

Layered Protocols

- **OSI 7-layer model was intended to be a foundation of a family of international standard protocols**
- **Those protocols *never gained much acceptance***
- **Roles of Session and Presentation layers are murky, at best.**
- **Most day-to-day protocols**
  - **work on a slightly modified layer system**
  - ***e.g. TCP/ IP uses a 4-layer rather than a 7-layer model***
- **Internet protocols (TCP/IP, *etc.*) are dominant**

TCP/IP Protocol Layers

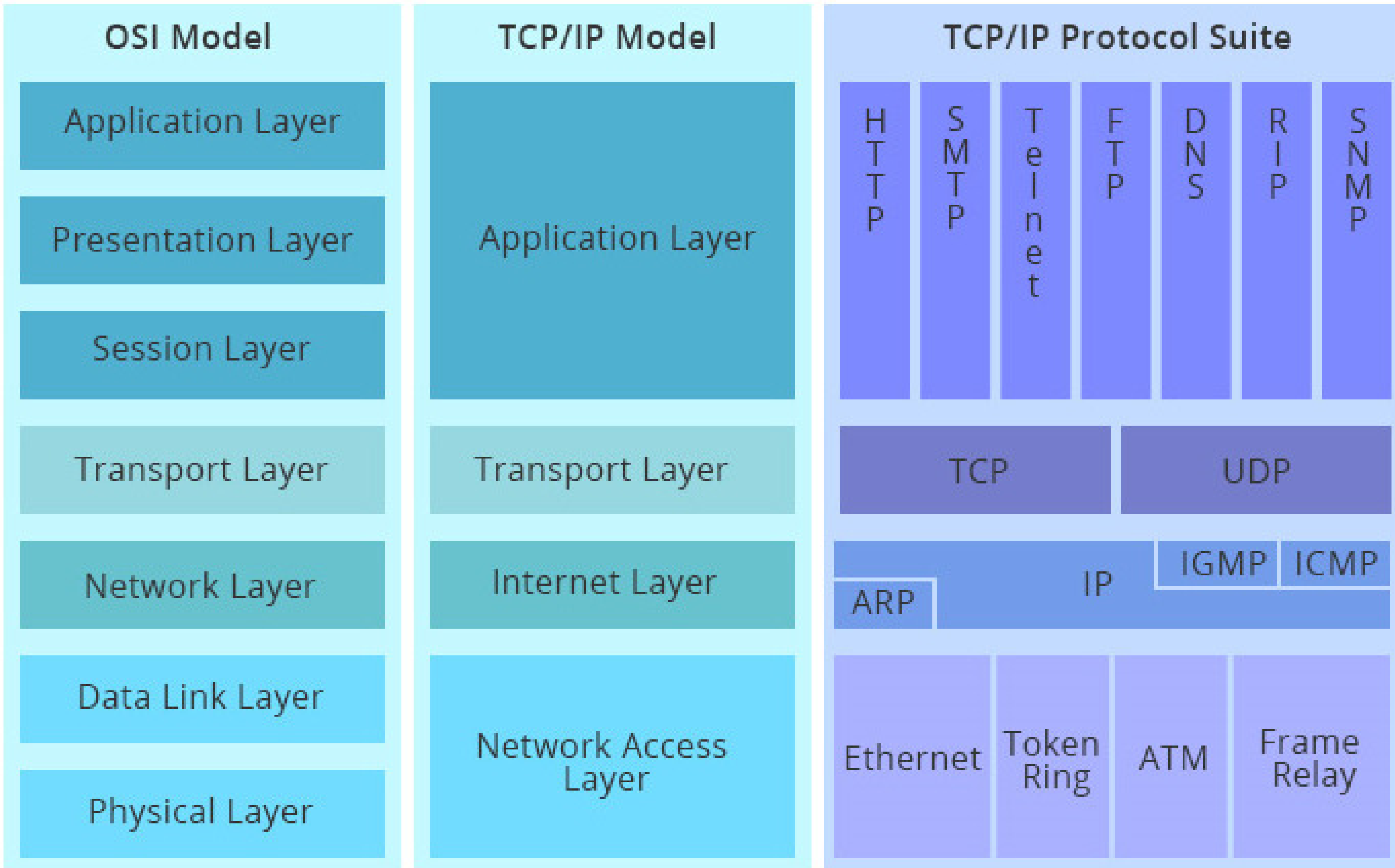


Image from https://community.fs.com/

Middleware Examples

- Authentication protocols
- Commit protocols for atomic transactions
- Multimedia protocols
- Remote procedure call protocols

Networking Computers

Principal Abstraction

Socket

- Originally created in BSD Unix (~1977)
- Subsequently, part of most operating systems
- Allows opening a connection ***between two processes across network***
- Connection:
  - a serial conversation between two end points
    - *e.g.*, processes, threads, tasks on different machines
  - organized as a sequence of messages or ***datagrams*** (or packets)
  - each connection distinct from all other connections

Some Terms

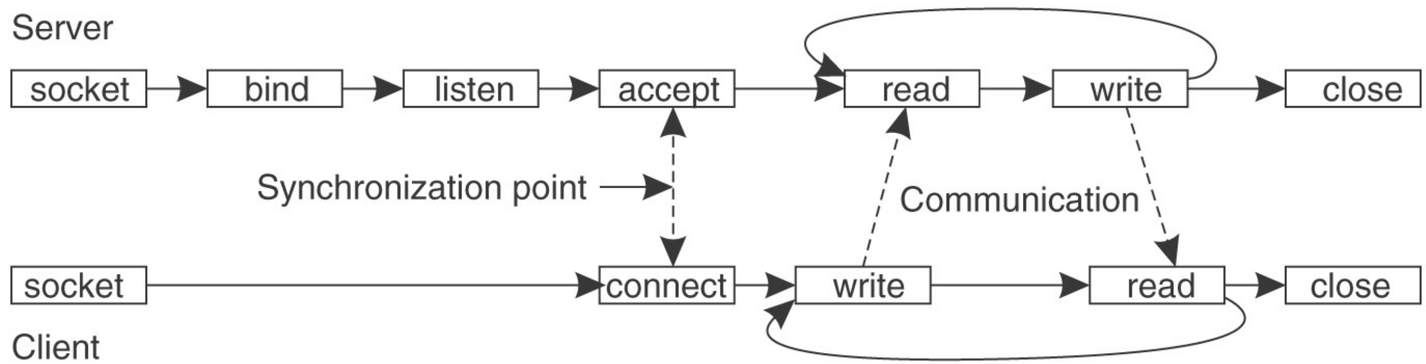
- **Port**:
  - A 16-bit number used within one computer to identify who/where to send packet to
- **Well-known port**:
  - A port with number < 1024, used by agreement for standard services (telnet, ftp, smtp, pop, *etc.*)
- **Registered port**:
  - A port with number 1024…49151, by agreement
- **Private port**:
  - A port with number > 49151
- **Socket**:
  - Comprises [IP Address: Port #]

Connection

- The *backbone of most message-oriented communication protocols*
- Each party retains knowledge of the other
- Each party retains information about state of the other (vis-à-vis the protocol itself)
- Each party "knows" if connection is broken

- Note: some of the popular protocols are "connection-less"
  - one side has ***no state information*** about the other side

Client-Server Connection



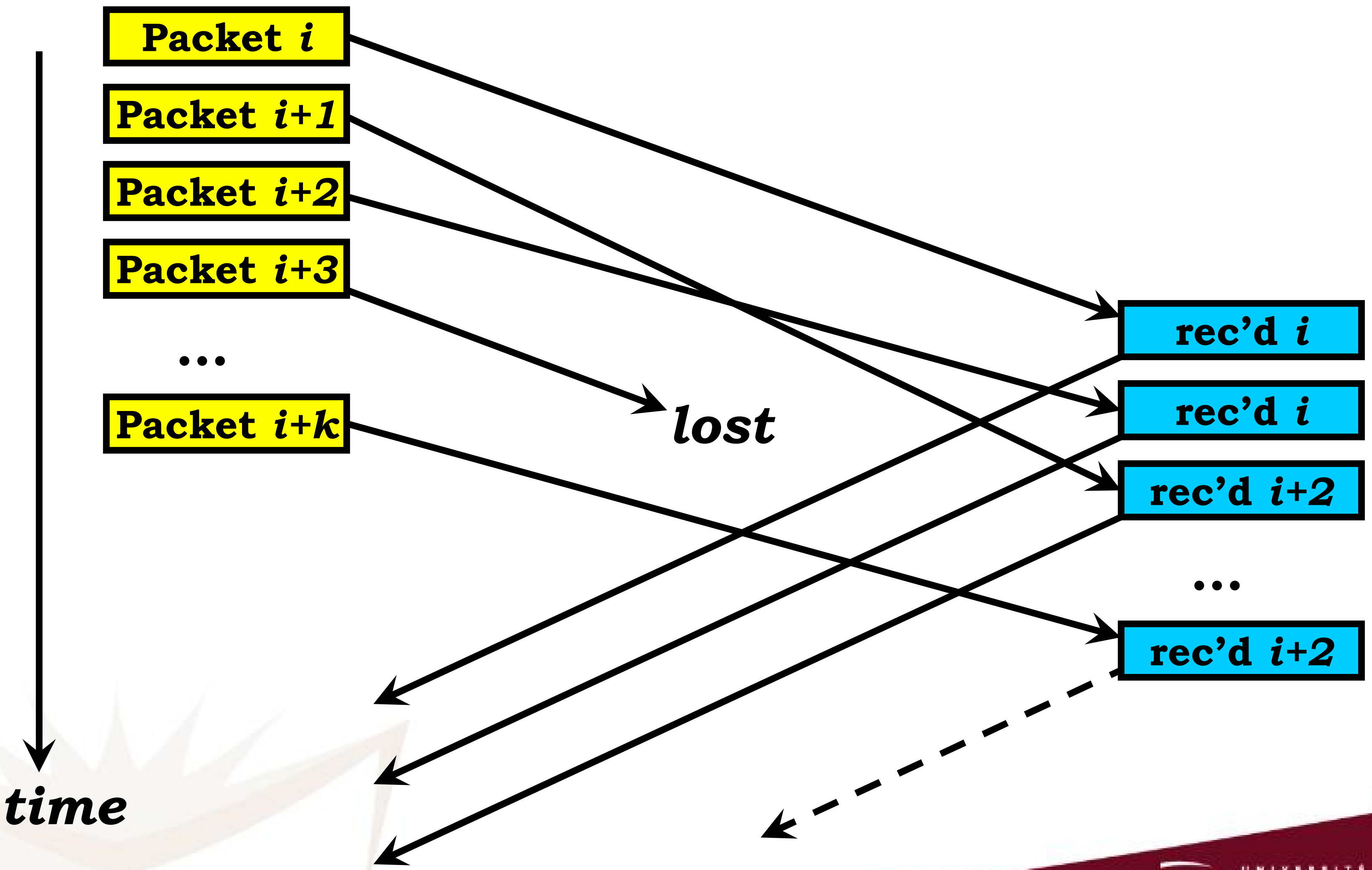
- **Process** (*e.g.*, game server) creates socket
  - On server side
- **Bind**
  - connect socket to port# (mostly well-known port)
- **Listen**
  - Sit and wait for a communication to come in
- **Accept**
  - Create new socket (bound to a private port) for purpose of responding to this caller (*e.g.*, client)

Reliable Connections

- ISO Transport layer partitions messages into packets
  - TCP – Transmission Control Protocol
  - Sequence number of current packet
  - Sequence number of last packet received correctly
- Receiver keeps track of sequence # of packets
  - *Reassembles in the correct order*
  - *Notify sender of missing, broken packets*
- Sender keeps copy of each packet until receipt acknowledged
  - Retransmits packets if no acknowledgement

Université Concordia University

Reliable Connections



Reliable Connections

- If ***acknowledgement received*** for packet i
  - **Delete from buffer** all packets ≤ i
- If ***no acknowledgement received*** within a reasonable time for packet k
  - **Retransmit from buffer** all packets ≥ k
- **Result**
  - **Recovers** from ***loss of packets***
  - **Recovers** from ***loss of acknowledgements***
  - **Works well** for ***reasonably reliable internet***
  - ***Doesn’t work so well for noisy, unreliable networks***

Packet Reception

- **How do we know if a packet is received correctly?**
- **Cyclic Redundancy Check (CRC)**
  - **Polynomial computed from packet header and body**
    - ***Usually 16 or 32 bits, computed by hardware***
  - **Appended to message**
  - **Recomputed on reception, *compared with transmitted CRC***
    - ***Equal $\Rightarrow$ packet received correctly***

Concordia University

TCP

- What Uses TCP?
  - Web browsers, FTP (mostly), Email, SSH, Telnet
  - Some MMOs, many real time strategy games, many turn based, less action oriented role playing games, and many end user applications
- Advantages of TCP
  - Simple to program
  - Reliable
  - Guaranteed ordering of information
  - Allows programmer to treat network like a file

TCP

- **Disadvantages of TCP**
  - **Slow in heavy network traffic**
  - **High latency if packets are *lost or received out of order***
  - **Programmer doesn't control packet size (problem if sending small packets, and *TCP sets a large minimum packet size*)**
  - **If a packet is lost or received out of order, TCP *waits for packet to be resent before allowing access to all packets*.**

Connection-less Communication

- **UDP – User Datagram Protocol**
  - **UDP is *not connection based*.**
  - **Used when**
    - ***a certain number of errors can be tolerated, and also***
    - ***where recovery from those errors is easy***
  - **Most packets will be delivered, but some may be lost.**
  - **There is no guarantee of the ordering of packets either.**

UDP

- What uses UDP?
  - Twitch-based games such as first person shooters (FPSs), action RPGs, Voice and Video Streaming, and ***programs where speed is more important than getting every packet***
- Advantages of UDP?
  - Low overhead
  - Fast, efficient transfers
  - Flexibility of packet size
  - Packets sent individually, ***only those that arrive are checked for integrity***

UDP

- **Disadvantages of UDP?**
  - **More difficult to program**
  - **Packet loss with no way to check**
  - ***Programmer responsible* for ordering packets**
  - ***Programmer responsible* for implementing acknowledgements, if required**

Connecting Multiplayer Games

- ***If all players are connected to the Internet***, **TCP/IP is normally used**.
- **It can be used to play with other players anywhere on the internet, assuming you know their IP address.**
- **Warning: Communication is getting more difficult now that people use *firewalls and routers*. These tend to *block messages and convert IP addresses*.**
- **On a local area network (LAN) *you can use it without providing addresses* (*e.g.*, broadcasting).**

Online Multiplayer Games

Multiplayer Games

- <u>Internet + wireless</u> making Multiplayer Computer Games more popular
- Most commercial computer games have a multiplayer option. *e.g.*, <u>with servers</u>:
  - Electronic Arts – Ultima Online (since 1997)
  - Blizzard – Battle.net (1996; Battle.net 2.0, 2009)
  - Microsoft's – MSN Games (1996)
- Consoles, too (PS5, Xbox One)
- Wireless devices, too (iOS/Android phones)
- Multiplayer games <u>may, or may not, be networked games</u>.

Université Concordia University

Postal Service Analogy

Image source: http://web.cs.wpi.edu/~imgd4000

Postal Service Analogy

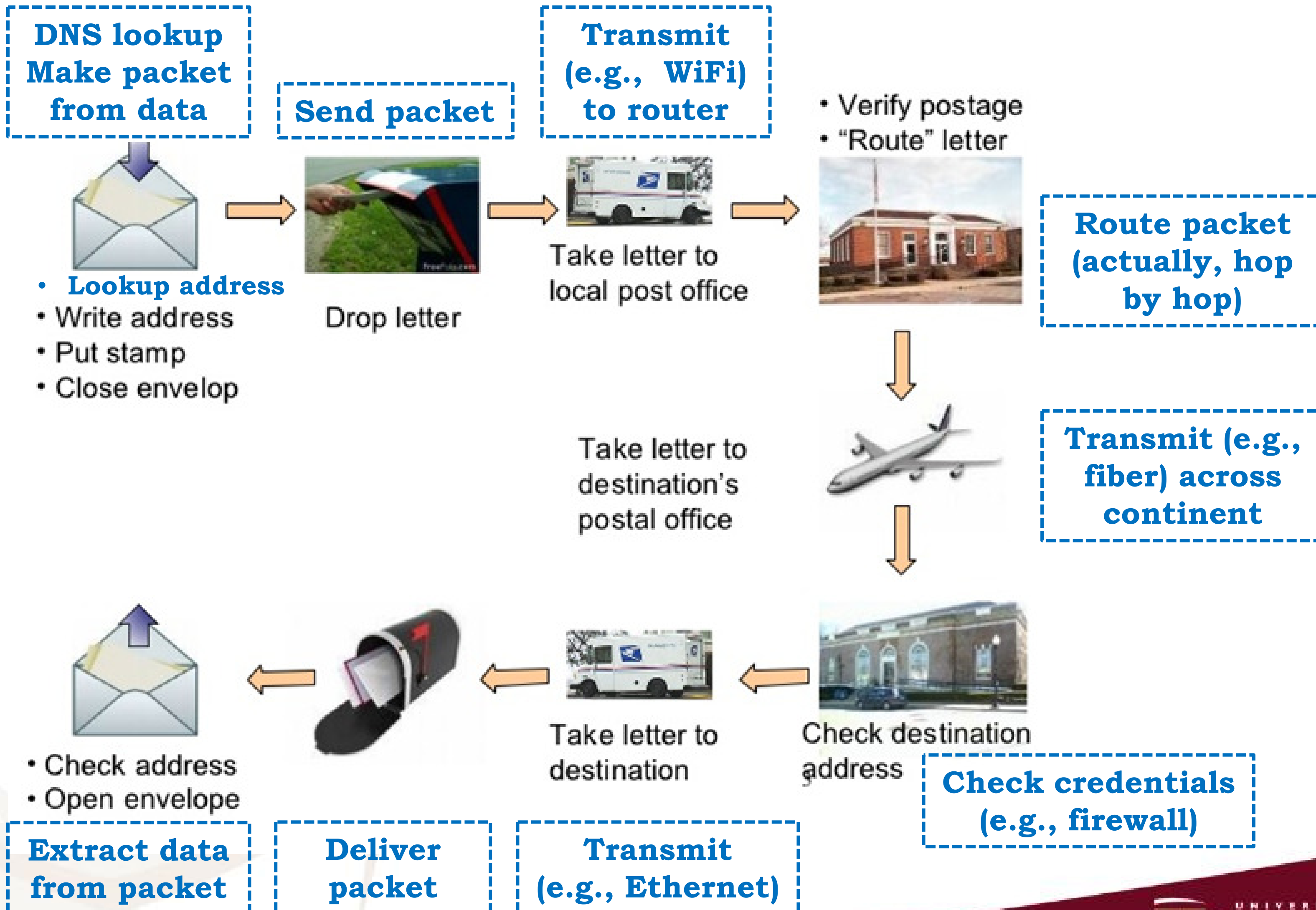


Image source: http://web.cs.wpi.edu/~imgd4000

Network Resources

- Distributed simulations face <u>three resource limitations</u>
  1. Network bandwidth
  2. Host processing power (to handle network)
  3. Network latency
- These are ***physical restrictions*** that the system cannot overcome
  - <u>Must be considered</u> in the <u>design</u> of the application
- (More on each, next)

Distribution Concepts

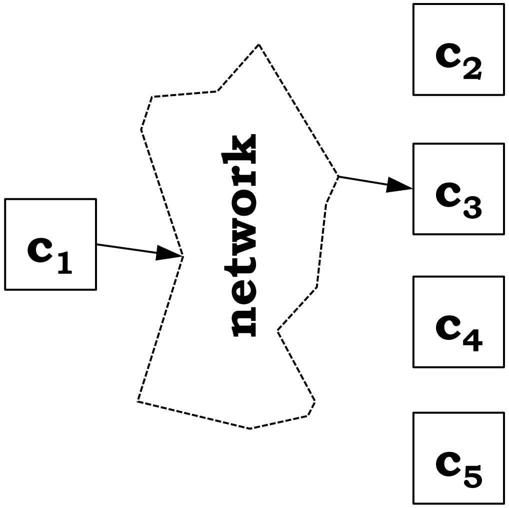
- Cannot do much about the resource limitations (bandwidth, lag, processing power)
- Should tackle problems at higher level
- Choose architectures for
  - Communication
  - Data
  - Control

1. Bandwidth (Bitrate)

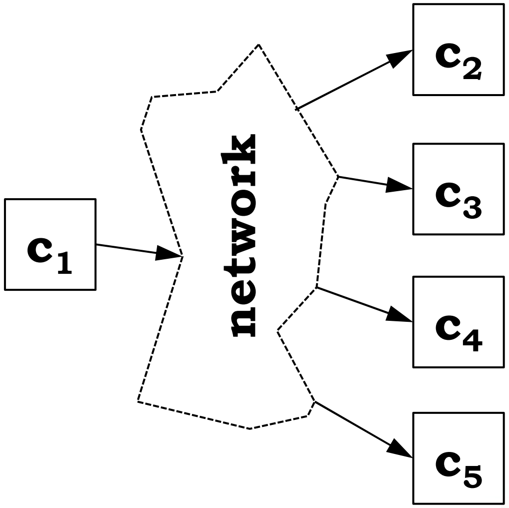
- Data sent/received per time
- LAN (Local Area Network) – 10 Mbps to 10 Gbps
  - Limited size and scope
- WANs (Wide A.N.s)– tens of kbps from modems, to 1.5 Mbps (T1, broadband), to 55 Mbps (T3)
  - Potentially enormous, Global in scope
- Number of users, size and frequency of messages determines bit-rate use
- As does transmission technique …

Transmission Techniques

- **Unicast, one send and one get**
  - **Static or DHCP (dynamically allocated IP address)**
  - ***Wastes bandwidth when path shared***

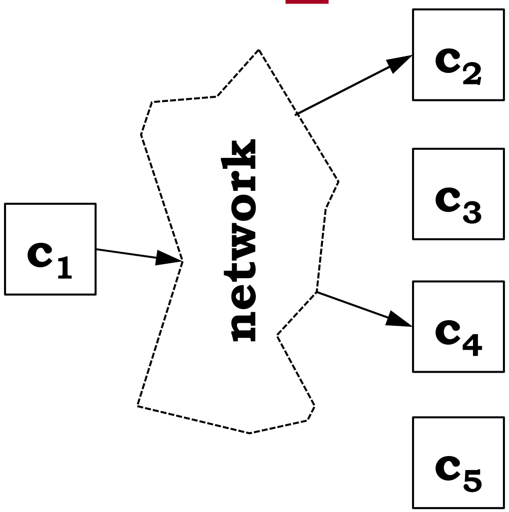


- **Broadcast, one send and all get**
  - **Not part of IPv6 (Multicast)**
  - **Perhaps OK for LAN since it's local and directed**
  - ***Wastes bandwidth when most don't need***

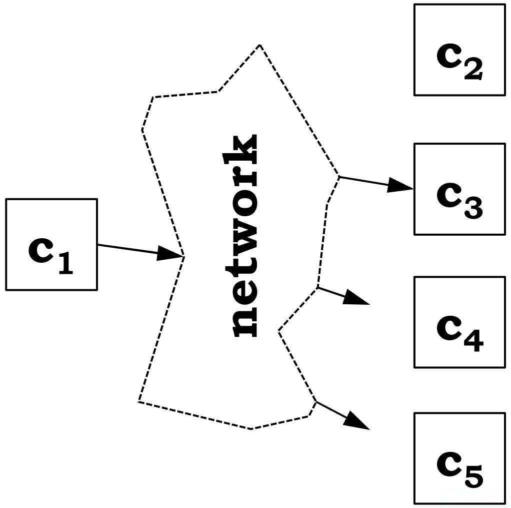


Transmission Techniques

- **Multicast, one send and only subscribed get**
  - *Requires a multicast capable router*
  - IPv4 does not support
  - Multicast overlay networks



- **Anycast**
  - New in IPv6
  - *One to One-of-many*
  - E.g., to closest out of a group of IP addresses



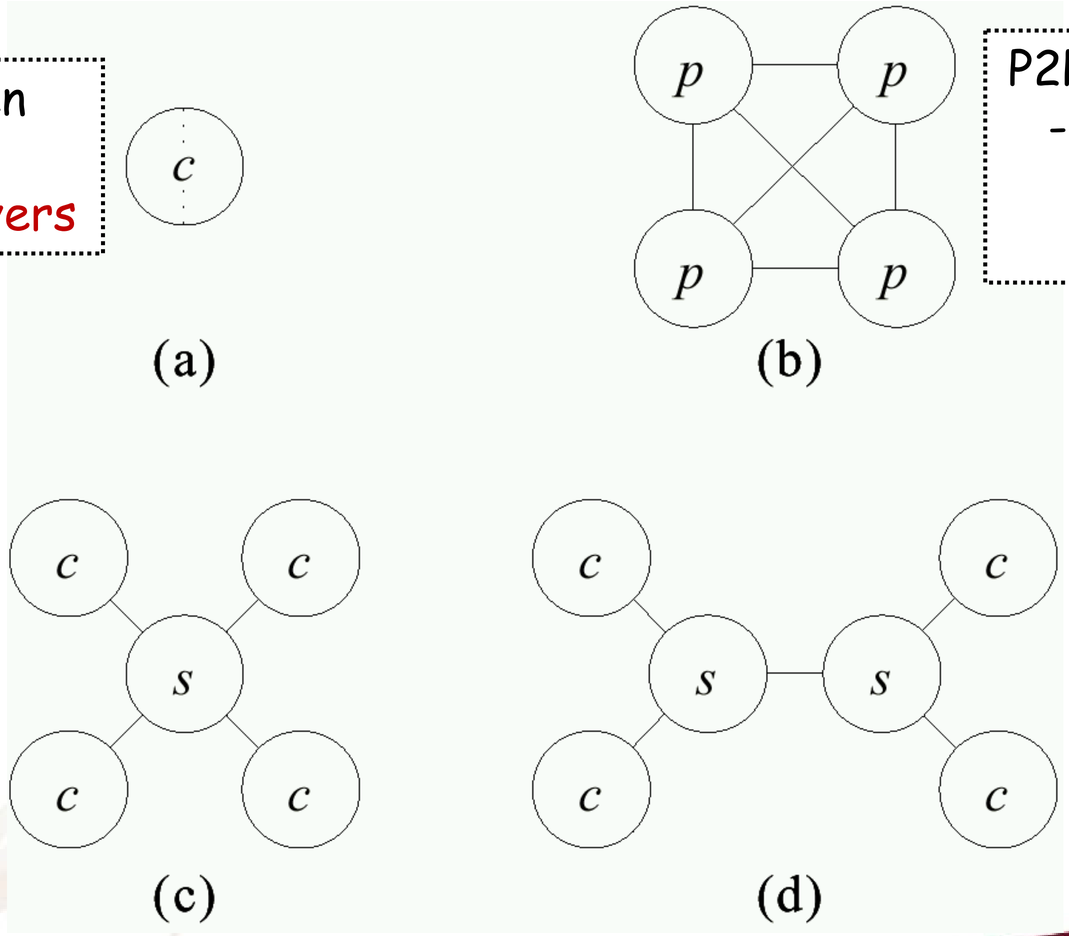
Real-Time Communications: Connection Models

- Broadcast
  - Good for player discovery (only) on LANs
- Peer to Peer
  - Good for 2 player games
- Client/Server
  - Good for 2+ player games
  - Dedicated lobby server great for player discovery

Communication Architecture

Split-screen Console
- Limited players

P2P: All peers equal
-Easy to extend
-Doesn't scale (LAN only)



Client/Server
One node server
- Clients only to server
-Server may be bottleneck

Server pool
-Improved scalability
-More complex

2. Computational Power

- Processing to send/receive packets
- Most devices powerful enough for raw sending
  - Can saturate a LAN
- Rather, application must process game state with each packet
  - Updating the game state whether using Input Reflection (*e.g.*, P2P) or State Reflection (*e.g.*, client-server)
- Especially critical on resource-constrained devices
  - *i.e.*- hand-held console, cell phone, PDA, *etc.*

3. Network Latency

- **Delay when message sent until received**
  - Variation (jitter) of delay also matters
- **Cannot be totally eliminated**
  - Speed of light propagation yields 25-30*ms* across Atlantic
  - With routing and queuing, usually 80*ms*
- **Application tolerances**:
  - File download – minutes
  - Web page download – up to 10 seconds
  - Interactive audio – 100s of *ms*

$$Lag = \frac{Distance\ (km)}{300}\, ms$$

3. Network Latency

- **Latencies tolerance in <u>multiplayer game</u> depends upon the game**
  - **First-Person Shooters – 100s of *ms***
  - **Real-Time Strategy – up to 1 second [SGB+03]**
  - **Other games**
- **Has effects on procedures like collision detection**
  - **Future collision predictions require knowledge of the exact game state at the current moment.**
  - **With latency, <u>*the current state is not always coherent and erroneous collisions may result*</u>.**

Compensatory Techniques

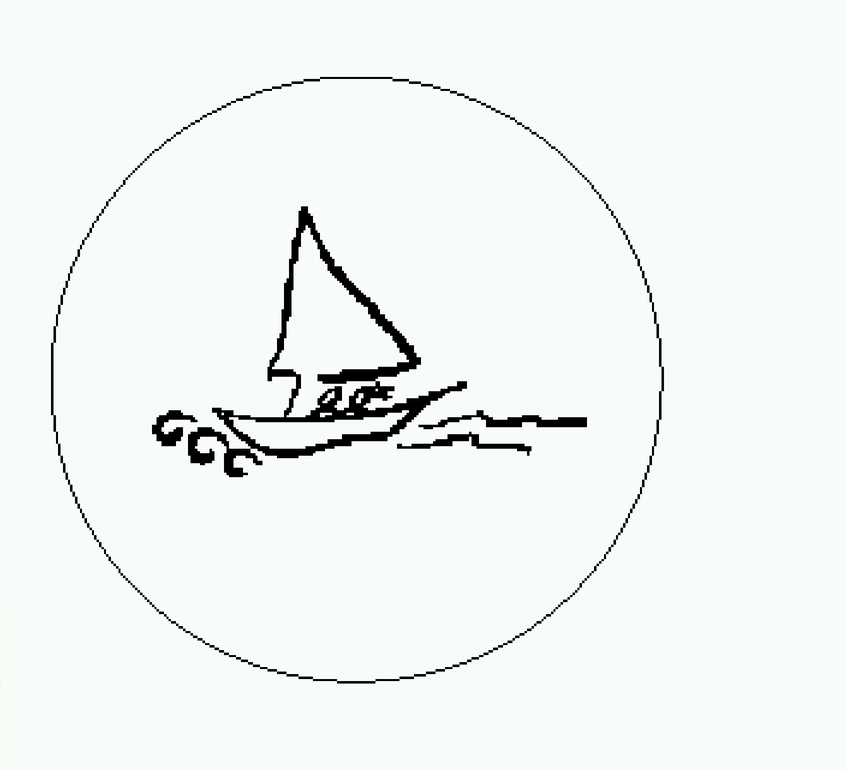
- <u>Architectures alone not enough</u>
- Must also design to compensate for latency and to maintain a synchronous playing environment
- Techniques (we'll look at these in detail):
  - Sending less or compressed data
  - Interest management
  - Dead reckoning
- Other Techniques (<u>next week</u>):
  - AI Assist

Message Aggregation

- **Sending less or compressed data**
  - **Lossless compression**: LZW algorithms for compression of stream data
  - **Opponent prediction**: Reduces the data sent
  - **Delta compression**: Send the differences not the whole data
  - **P2P**: Avoids sending irrelevant data to the server
  - **Update Aggregation**: Collect data for a client (or the server for inputs from client) and sent all the collected data at a time instead of sending instant update data

Interest Management – Auras

- **Nodes express area of interest to them**
  - **Send information of close objects (don't send distant objects' information)**



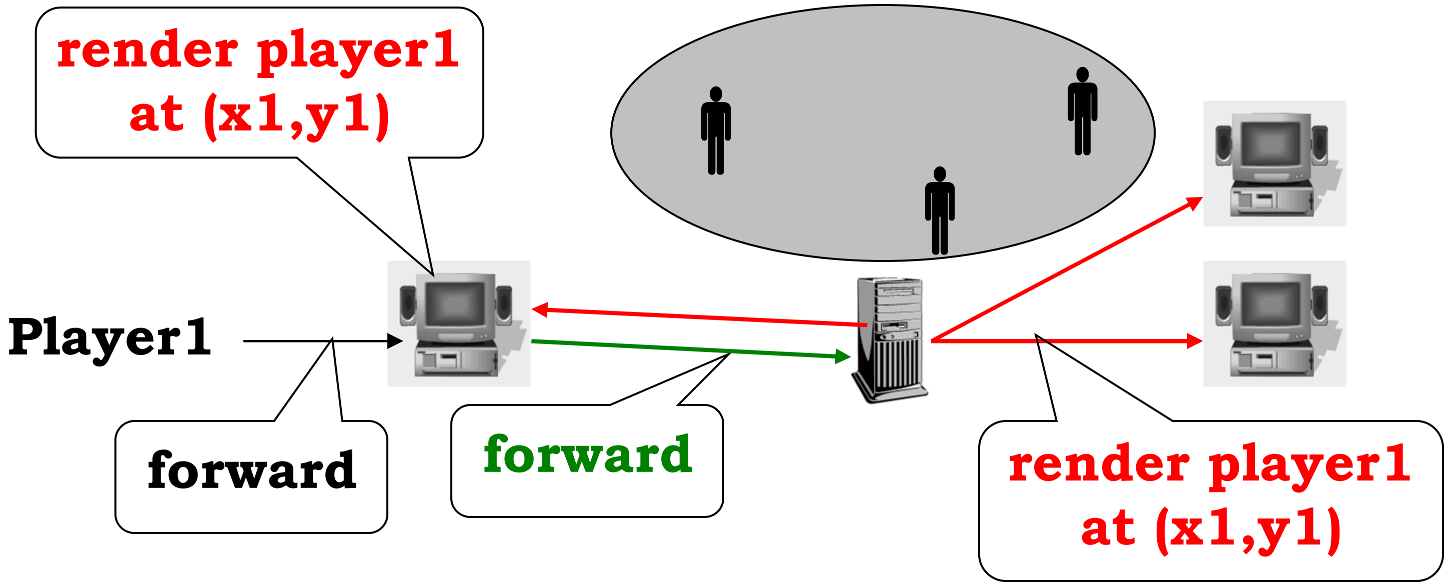
- **Only circle sent even if world is larger.**
- **But *implementation is complex***

Massively Multiplayer Online Games (MMOG)

- Many games of the RPG genre already allow thousands of users to concurrently participate in a single game session.
- There are important genres, in particular action and strategy games, which have not been scaled to the massively multiplayer realm so far.
  - These games have *hard requirements in terms of scalability*, in particular regarding density: *many players tend to congregate in small locations*.

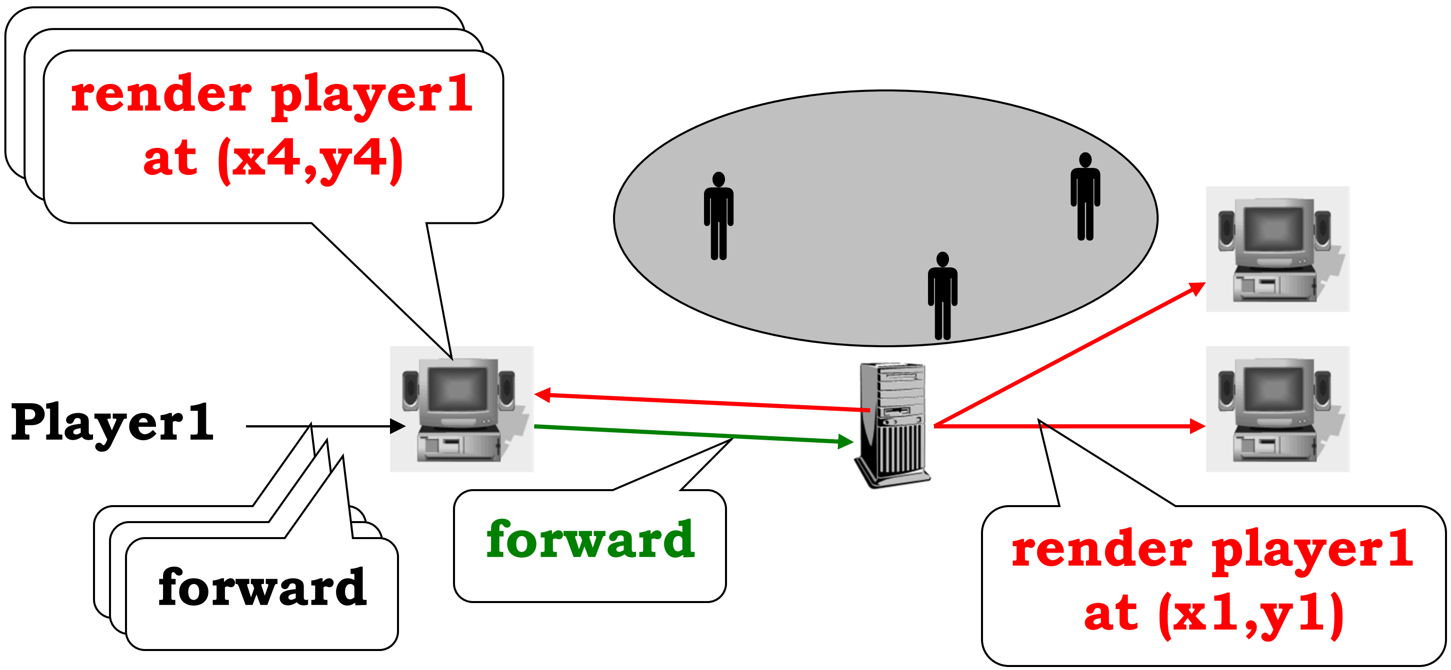
Latency Compensation

- Naïve approach: dumb client



Response time for player:
round-trip to server + server processing

Predicting Where I Am

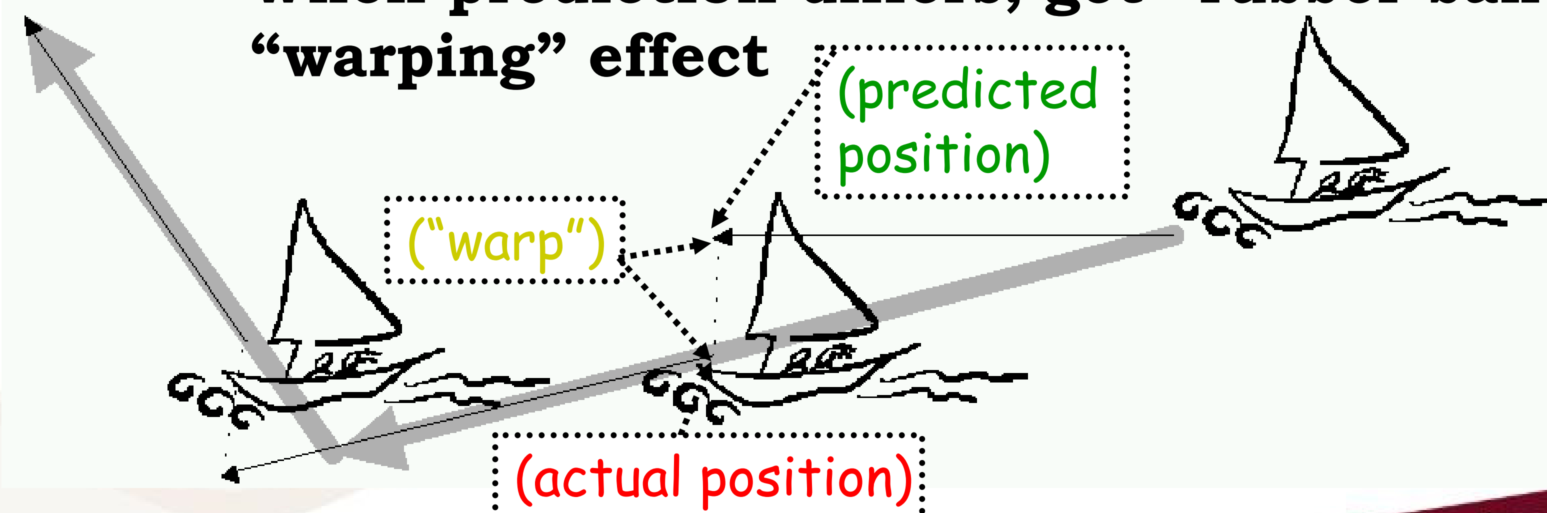


Predicting where you are

- Updates about ***other players' locations not continuous***
- Two possible compensations:
- **Extrapolation** (dead reckoning)
  - At last update, player2 is at (x_1, y_1) facing N with speed S
    ➔ Predict that they should be at (x_2, y_2) now
  - **Not good**: in FPS, player movement is frequently very non-deterministic
- **Interpolation** (lag compensation)
  - Impose an "interpolation delay" for rendering
  - Or use delayed positions for physics.

Dead Reckoning (Extrapolation)

- **Client-side lag compensation**
  - **Based on ocean navigation techniques**
  - ***Predict position based on last known position plus direction***
    - **Can also only send updates when they deviate past a threshold**
  - **When prediction differs, get “rubber-banding“ or “warping” effect**



Lag Compensation
(Interpolation)

- **Interpolation introduces a fixed lag (int. delay)**
  - ***e.g.*, always see where you were 100*ms* ago**
  - **Need to lead the target when aiming**
  - **Require players to extrapolate! So, instead use:**
- **Server-side lag compensation**
  - **Render current scene**
  - **Server uses the old location to compute hit/miss**
  - **Allows natural aiming/shooting**
  - **Possible weird experiences for players being fired upon**

    ➔ trade-off for better game play

Data and Control Architectures

- Want consistency
  - Same game state on each node
  - Needs tightly coupled, low latency, small nodes
- Want responsiveness
  - More computation locally to reduce network load
  - Loosely coupled
- In general, there is a trade-off between the responsiveness of the distributed state processing and the degree of consistency.

Multiplayer Architectures

- **Centralized** (*e.g.*, client-server)
  - Server node holds data so view is consistent at all times
  - Lacks responsiveness (clients must be updated)
- **Distributed and Replicated** (*e.g.*, peer-to-peer)
  - Replicated has copies, used when predictable
    - *e.g.*, each peer computes an NPC's position.
  - Distributed has local node only, used when unpredictable (*i.e.*, players)
    - *e.g.*, each peer shares non-predictable game state info such as player input.
  - Lacks coherence

Security

Security and Cheating

- **Unique to games**
  - **Other multi-person applications don't have**
  - **In comparison, military simulations are not public and are considered trustworthy**
- **Main security goals:**
  - **Protect sensitive information (*e.g.*, credit cards)**
  - **Maintain level playing field**
- **Cheaters want:**
  - **Vandalism – create havoc (relatively few)**
  - **Dominance – gain advantage (more)**

Packet and Traffic Tampering

- **Reflex augmentation** - enhance cheater's reactions
  - Example: aiming proxy monitors opponents movement packets, when cheater fires, improve aim
- **Packet interception** – prevent some packets from reaching cheater
  - Example: suppress damage packets, so cheater is invulnerable
- **Packet replay** – repeat event over for added advantage
  - Example: multiple bullets or rockets if otherwise limited

Université Concordia University

Preventing Tampering

- Cheaters figure out by changing bytes and observing effects
  - Prevent by MD5 checksums (fast, public)
- Still cheaters can:
  - Reverse engineer checksums
  - Attack with packet replay
- So:
  - Encrypt packets
  - Add sequence numbers (or *encoded sequence numbers*) to prevent replay

Encryption Methods

- **Keyed**
  - Public Key (Asymmetric — Key Pairs)
  - Secret Key (Symmetric — Same Key)
  - Ciphers
    - Block and stream ciphers
- **Message Digest**
  - produces a checksum to verify message integrity
- **Certificates**
  - aka digital IDs — authentication through a trusted third party [VeriSign]
- **IPSec**

Encryption

- **Popular Packet Encryption Algorithms**
  - **The most popular choices for packet encryption are**
    - **RC4 (Rivest Cipher 4)**
    - **RC5 (Rivest Cipher 5)**
    - **RC6 (Rivest Cipher 6)**
    - **Or one of the algorithms from the Advanced Encryption Standard (AES)**
  - **Also people *use their own encryption method* but this is not always the smartest thing to do because it *may be easier to crack than one of the famous ones*.**

Encryption Issues

- No matter what kind of encryption that is used, the decryption of the data from the server must be done with the client and this ***creates possible issues***.
- Some of the issues that come from this are:
  - Allows for cheating, through packet sniffing.
  - Allows the user to ***modify the data within the packet***.
  - Allows the ***packets to be decrypted and then the info turned into a private server***, for example, for games like World of Warcraft.

Information Exposure

- **Decryption on the client side *allows cheater to gain access to replicated, hidden game data* (*i.e.* status of other players)**
  - **Passive, since does not alter traffic**
  - **Examples: defeat "fog of war" in RTS; *see through walls in FPS***
- **Cannot be defeated by network alone …**

Information Exposure

- Instead:
  - Sensitive data should be encoded
  - Kept in <u>hard-to-detect memory location</u>
  - Centralized server may detect cheating (example: attack enemy could not have seen)
    - Harder in replicated system (e.g., P2P), but can still share

Université Concordia University

Using Unity

Networked Multiplayer Games

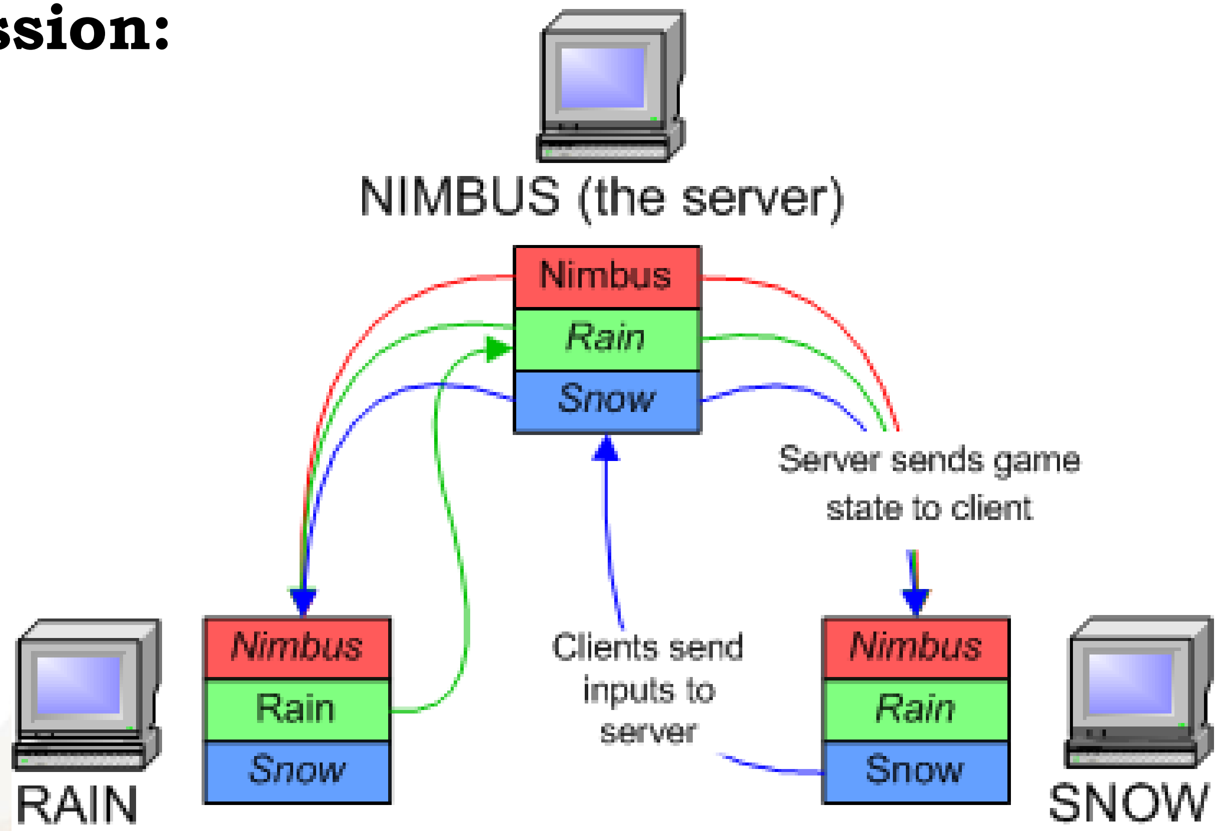
- Two types of network topologies are supported by Unity for Multiplayer Games (*i.e.*, networked game simulation):
  - Client/Server
  - Peer-to-Peer (P2P)
- Two Transport Layer protocols:
  - UDP for generic communications
    - Can configure "quality of service" to include reliability.
  - WebSockets (based on TCP/UDP) for WebGL

Authoritative Server

- The ***server to perform all world simulation***, **application of game rules and processing of input from the player clients**.
- **Each client sends their input (in the form of keystrokes or requested actions) to the server and *continuously receives the current state of the game from the server*.**
- **The client never makes any changes to the game state itself.**

Authoritative Server

- This figure shows the data flow for a client/server game with three computers in the session:



Non-Authoritative Server

- A non-authoritative server exists ***only as a kind of proxy between all the connected clients***.
- Server relays messages sent by clients and ***doesn't know anything or very little about the game logic***.
- All of the **game logic is implemented on a client**.
- They are the owners of their objects and are the ***only agents permitted to send local modifications*** of those objects over the network.

Non-Authoritative Vs Authoritative Server

Non-authoritative server

- **Cons**
  - **Prone to hacking and cheating, since it is <u>possible to change original game logic on client</u>**
  - **Hacking software even exists that can <u>automatically search and modify important game variables like</u> lives, score, *etc*...**
- **Pros**
  - **Since all the logic is on the client side, the <u>*server requires much less CPU and memory*</u> resources than an authoritative server.**
  - **<u>*Less time for the messages to travel over the network*</u> compared to an authoritative server.**

Université Concordia University

Semi-authoritative Server

- Semi-authoritative server is a blend between the two aforementioned approaches, giving some authority to the client over certain aspects of game logic.
  - For example, in semi-authoritative setup, ***client reports to the server when an opponent is hit*** and should receive damage, and the server keeps track of a player's health status and decreases it accordingly.

Networking Options

- **Unity Networking**
  - Easy to use, but limited scalability
  - Works well for rapid prototyping
  - UNet no longer supported
- **Photon by ExitGames**
  - Not as easy to use
  - Not aware of Unity geometry
- **uLink** (UnityPark Suite) by MuchDifferent ($$)
  - More similar to Unity Networking
- Others: SlimNet, SmartFox, *etc.*

Unity Networking

Network Communication

Remote Procedure Calls (RPCs)

- **Used to invoke functions on other computers across the network**
  - **"network" can mean the message channel between the client and server when they are both running on the same computer.**
- **Clients can send RPCs to the server, and the server can send RPCs to one or more clients.**
- **Most commonly, they are used for actions that happen infrequently.**
- **They are used for managing and executing individual events.**

Université Concordia University

Unity Networking

Network Communication

State Synchronization

- Used to share data that is constantly changing.
  - The best example of this would be a *player's position in an action game.* By constantly relaying data about this player's position, the game can accurately represent that position to the other players.
- This kind of data is regularly and frequently sent across the network. Since this *data is time-sensitive*, it is important to reduce the amount of data that is sent as far as possible.

Université Concordia University

Unity Networking

Connecting servers and clients

- **Connecting Private addresses**
  - **Private addresses are IP addresses which are not accessible directly from the Internet. *e.g.*, a local IP address given by a router.**
  - ***A relay server can be used*.**
- **Connecting Public addresses**
  - **Straightforward**
- **Connecting Public addresses behind an external firewall**
  - ***A relay server can be used*.**

Unity Networking

Connecting servers and clients

- **Matchmaking service** (see unity3d.com)
  - Need to register the project first in Service Window's Multiplayer panel
  - In Matchmaking service, can create games, get lists of active games and join and leave games
  - To use it, derive a script from special script NetworkMatch and attach it to a manager object
  - Facilitates using a relay server
- NAT Punchthrough is no longer supported as of Unity 5.x; may come back in future in some form due to latency considerations

Unity Networking

Connecting servers and clients

- **Relay Server**
  - **Works closely with the matchmaker server**
  - **Network traffic goes through a relay server hosted by Unity in the cloud instead of directly between the clients.**
  - **To use a relay server instead of a direct connection, you must populate the singleton NetworkMatch.matchSingleton (to ensure right relay server is used for the match).**
  - **The *higher level classes handle the relay automatically***

Unity Networking

Multiplayer Games

- *Unity's networking is integrated into the engine and the editor*. **Provides:**
  - **A NetworkIdentity component for networked objects.**
  - **A NetworkBehaviour for networked scripts.**
  - **Configurable automatic synchronization of object transforms.**
  - **Automatic synchronization of script variables.**
  - **Support for placing networked objects in Unity scenes.**
  - **Network components: NetworkAnimator, NetworkServer, NetworkClient, etc.**

References/Resources

- Introduction to Networks, CS-4513 Distributed Computing Systems, WPI (2007) [PPT slides]
- Aspects of Networking in Multiplayer Computer Games, by J. Smed, T. Kaukoranta and H. Hakonen, The Electronic Library, Volume 20, Number 2, Pages 87-97, 2002 [Paper & PPT slides]
- Using TCP vs UDP in networked games by Daniel Loftis for CS594 Networked Games, UTK [Webpage]
- Encryption for networked games by Joshua Strange for CS594 Networked Games, UTK [Webpage]
- Tutorial 5: Adding Multiplayer and Networking Support to the Game, MSDN [Webpage]
- Networking and Online Games by G. Armitage, M. Claypool, and P. Branch. West Sussex: John Wiley & Sons Ltd. (2006) [Text]
- Ch 5.6 Network and Multiplayer in Introduction to Game Development, edited by Steve Rabin (2005) [Text]
- Multiplayer Online Games, by An-Cheng Huang and Bruce Maggs [PPT]
- Multiplayer Online Games, by Umut Riza Erturk [PPT]

References/Resources

- Network Architecture: Client/Server, MSDN App Hub, Microsoft: http://create.msdn.com/en-US/education/catalog/sample/network_cs [Webpage]
- Game Maker Tutorial : Creating Multiplayer Games, by Mark Overmars [PDF]
- Jens Müller, Sergei Gorlatch, Tobias Schröter, and Stefan Fischer. 2007. Scaling multiplayer online games using proxy-server replication: a case study of Quake 2. In Proceedings of the 16th international symposium on High performance distributed computing (HPDC '07). ACM, New York, NY, USA, 219-220. [PDF]
- [SGB+03] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The Effect of Latency on User Performance in Warcraft III. In Proceedings of the ACM NetGames Workshop, May 2003 [Paper]
- Unity Networking Documentation, http://docs.unity3d.com/Manual/UNetInternetServicesOverview.html [Webpage]