Networking for Games

IMGD 4000

Introduction (1 of 2)

- Games are increasingly networked
  - Multi-player, connecting PCs and Game consoles (e.g., Counter-strike, Halo)
  - Single-player, pulling and pushing content to Web service (e.g., Lumosity)
- Emerging services play game in "cloud", sending rendered game down as video
  - (However, will not talk about this approach much)
- All require an understanding of networking (conversant), with enough knowledge to design and build network game (develop)

Introduction (2 of 2)

- For now, "networking" mostly means "Internet networking", so that will be our reference
- Other networking aspects that can be relevant for games includes:
  - Ad Hoc / Mesh networking
  - Short-range wireless (e.g., Bluetooth)
  - Network security (including cheating)
  - Mobile application (game) development (often networked)
- These, and other topics available in-depth from your friendly, neighborhood WPI course

(next slide)

Networking at WPI

- General, core networks:
  - CS 3516 – Computer Networks
    - Broad view of computer networks, top-down
  - CS 4516 – Advanced Computer Networks
    - In-depth computer networks, more "under the hood"
- Networks applied to specific domains
  - CS 4513 – Distributed Systems
  - CS 4518 – Mobile and Ubiquitous Computing
  - CS 4241 – Webware: Computational Technology for Network Information Systems
  - CS 4404 – Tools and Techniques in Computer Network Security
- Also grad courses
  - CS 513 – Introduction to Local and Wide Area Networks
  - CS 528 – Mobile and Ubiquitous Computing
  - CS 529 – Multimedia Networking
  - CS 530 – High-Performance Networks
  - CS 533 – Modeling and Performance Evaluation of Network and Computer Systems
  - CS 558 – Computer Network Security
  - CS 577 – Advanced Computer and Communications Networks

This deck → core networking applied to computer games.

The Internet from the Edge (1 of 2)

- Reasonable analogy → Postal Service
  - Letter in envelope
  - Find address of business in phonebook
  - Address on envelope
  - Put in Mailbox → *trust* that will reach destination
  - Don't know how envelope gets there
  - Delivery takes different amounts of time for each
    - Generally, further away longer (but not always)
  - Use external ways to confirm
    - (ex: Use phone, or resend letter until confirmation)

The Internet from the Edge (2 of 2)

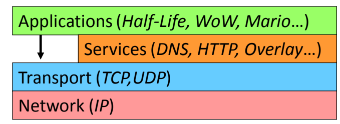
- Reasonable analogy → Postal Service
- Users view the Internet similarly → An opaque cloud
  - User application content is like *letter*
  - Putting content into packet for sending is like *envelope*
  - IP address is like *address* on envelope
  - Internet name service (DNS) to find address is like *phonebook*
  - Internet routing is like mail carrier delivery route
  - If don't get return letter, resend (packet) to make sure message is received
    - There are other ways to ensure delivery (e.g., repair) that we will not discuss

Outline

- Introduction (done)
- Basic Internet Architecture (next)
- Loss and Latency
- Latency Compensation Techniques
- Client-Server Synchronization

The Internet

- Many design decisions and end-user experiences for multi-player networked games derive from nature of Internet
  - "Best Effort" service
  - Internet naming and addressing
  - Transport protocols (TCP/UDP)
- Layered



Internet Provides "Best Effort" Service

- Few guarantees on timeliness
  - Can take milliseconds, 100's of milliseconds, or even seconds to deliver packet
- Few guarantees on arrival certainty
  - Sometimes packet doesn't arrive (loss)
  - Or arrives out of order (e.g., packet #3 arrives before packet #2)
  - Or can arrive twice (duplicates, uncommon but possible)
- Time to reach destination called latency
  - Lag typically latency + end-host (server and client) time
    - Often, players have hard time distinguishing

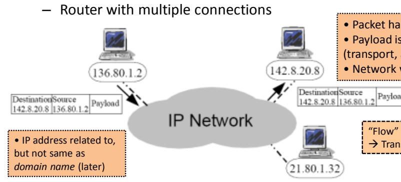
(More on loss and latency later)

Endpoints and Addressing

- IPv4 numerical 32-bit (4 byte) values
  - Dotted quad form: `192.168.1.5` or `130.215.36.142`
  - In theory, 2^{32} (about 4 billion) addresses, but practically fewer since allocated in blocks
- Each Internet host has IP address
  - Client running game client
  - Server running game host
- Some have 2
  - Client with wireless and wired network (multi-homed)
  - Router with multiple connections

IPv6 has 2^{128} addresses (enough for 100 addresses for each atom on the earth surface), but not widely deployed in U.S.

- Packet has: source, destination
- Payload is upper layer (transport, application)
- Network worries about arrival



- IP address related to, but not same as *domain name* (later)

"Flow" determined by *port*, too
→ Transport layer (next)

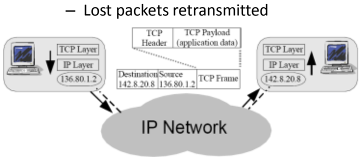
Transmission Control Protocol (TCP)

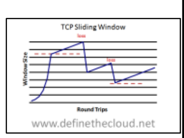
- Many applications sensitive to loss, not time
  - e.g., File transfer (.exe), email
  - Need reliable, ordered transfer of bytes
- Frames data → send as IP packets
- Provides connection
- Uses *window* for outstanding packets
  - Provides *flow control* and *congestion control*
  - Window grows with success, shrinks with loss
  - Lost packets retransmitted

Some games more sensitive to time
→Don't use TCP

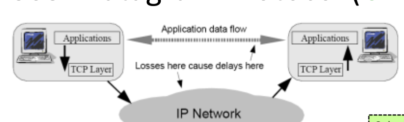
But many games do use TCP
→RTS, MMO

Generally, easier to use TCP for games!





User Datagram Protocol (UDP)

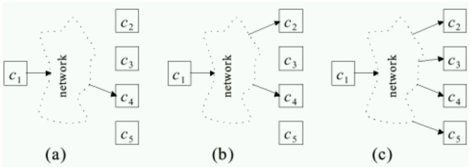


- Some applications sensitive to time
  - e.g., Voice over IP (VoIP)
  - Some games (e.g., First Person Shooter)
- Unreliable, connectionless
- No flow control (sender can go faster than receiver)
- No congestion control (sender can go faster than network)
  - Note: IP does ensure there are no bit errors (via Cyclic Redundancy Check, CRC)
- *Lightweight*, but *application must handle loss!*

Only use UDP if you know your application is sensitive to latency!

Otherwise, use TCP (see previous slide)

Unicast, Multicast, Broadcast



(a) Unicast, one send and one receive
- Wastes capacity when path shared

(c) Broadcast, one send and all receive
- Perhaps ok for LAN, but cannot do on Internet
- Wastes capacity when most don't need

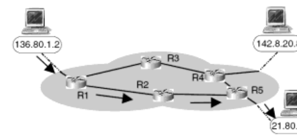
(b) Multicast, one send and only subscribed receive
- Current Internet does not support
- Multicast can work for overlay networks (separate topic)

Note, UE4 provides a multicast networking feature. However, this is *not* true IP multicast, but rather replicated unicast to all clients.

Slide 13

Connectivity

- Often edge most important
  - Game developer does not see internals
- But some aspects important for understanding network performance
  - Hierarchy
  - Routing
  - Link-layer

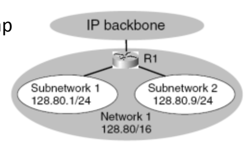


Independent choice for packet route based *solely* on destination address
- Not based on sender
- Not based on QoS

Slide 14

Connectivity – Hierarchy

- Routers designed for speed
  - Get packet to outgoing link asap
- Value + Prefix size
  - `128.80.0.0/16` → all w/`128.80` go to R1
  - R1 forwards more precisely to subnet
  - WPI has `130.215` with
    - `130.215.28` CS subnet
    - `130.215.36` CCC subnet (CCC1, …)
    - `130.215.16` ECE subnet…

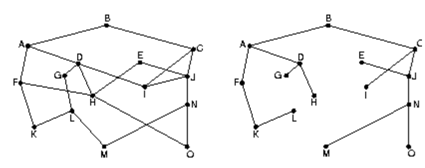


Slide 15

Connectivity – Routing

- Routers use dynamic routing
  - Discover topology
  - Pick "best" routes (want tree)
    - Typically shortest path (# hops, latency…)

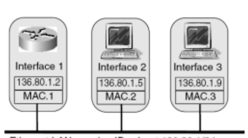
Note: Local (internal to ISP) routing protocol different than among ISPs (ASes). The "cost" between ASes different than simply distance.



Slide 16

Connectivity – Link Layer

- Link layer conveys packets across LAN
  - Medium Access Control (MAC)
- IP address mapped to data link layer
  - Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11)
  - MAC address 48-bit. E.g., `00:0F:1F:81:41:6C`
  - MAC address specified by vendor on card
- IP to MAC assignment:
  - Fixed (e.g., register computer with netops)
  - Dynamic (assigned when boot)



Typically, network game won't care about link layer since usually fast. But …. wireless, particularly wide-area wireless (e.g, 4G) can have 10s or 100s and even 1000s of millseconds of latency!

Slide 17

Miscellaneous

- Time-to-Live
  - Prevent loops (routers may have different shortest-path trees)
  - 8-bit value (0 to 255)
  - Decrement by one each hop
  - If zero, then discard
- Maximum Transmission Unit (MTU)
  - IP packet could be 64 KBytes
  - In practice, bound by Ethernet (prevalent standard)

  → 1500 byte payload, so 1460 bytes for application payload
  - If larger, then fragment into multiple IP packets
    - Re-assemble at end
    - If one lost, all lost!
- First Hop
  - Only know egress (e.g., first router)

`ifconfig` (Linux)
`ipconfig /all` (Windows)

For network games using UDP, keep data payloads smaller than MTU!

Slide 18

Address Management Mini-Outline

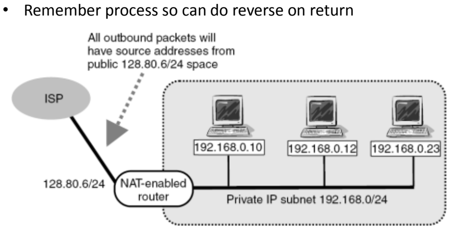
- Network Address Translation
- Dynamic Host Configuration Protocol
- Dynamic Name Service

Network Address Translation (NAT) (1 of 2)

- Used at boundary of ISP
  - Where internal private addresses use external publicly routable address
- Good if internal address not allocated
  - Ex: private networks
    - `10/8, 172.16/12, 192.168/16`
- Also, may help keep internal network secure (but not sufficient)

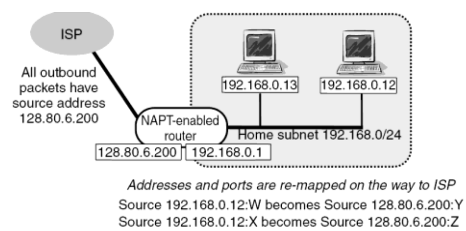
Network Address Translation (NAT) (2 of 2)

- Source hosts use private IP
- Forward to NAT router
- Swap private source address with public address (could be range)
  - 1-to-1 mapping between source hosts and public addresses
- Send to ISP for Internet routing
- Remember process so can do reverse on return



Network Address Port Translation (NAPT) (1 of 2)

- Have only 1 public address for multiple private addresses



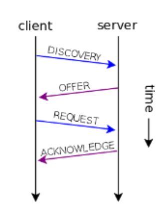
Network Address Port Translation (NAPT) (2 of 2)

- Good:
  - Easy to renumber (one number)
  - Only need one public address
- Bad: Breaks transparency (need to add functionality for each new protocol)
- Hard for outside hosts to access inside
  - Ex: what if two different *Unreal Tournament* servers inside?
  - Need non-standard ports that *clients* know about
    - Typically, local server register w/master server
      - Gives IP adress + Port where server is
  - Need to configure NAT box to forward ports

For network game, cannot rely upon reaching server behind NAT!

Dynamic Host Configuration Protocol (DHCP)

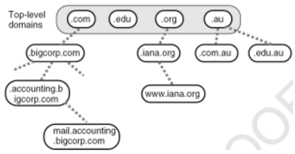
- Hosts need: IP address, subnet mask, IP address of at least one router
  - Use DHCP to get from LAN device
    - Typically with WLAN router, cable modem, …
- Client broadcasts DHCP discovery to port 67
  - Identifies its MAC address (e.g., 00:0a:95:9d:68:16)
- DHCP server responds w/IP + Mask + Router IP
- Client confirms, may request additional information (could be more than one DHCP server)
- Server ACKs



For network game, host may not have same IP address each time!

Domain Name System

- Map text names to IP address
  - Ex: `www.wpi.edu` mapped to `130.215.36.26`
  - Names more human-readable
- Minimal `<name>.tld` (top-level-domain)
  - tld: `.com, .gov, .edu`
  - tld: `.au, .fr, .uk`



- Hierarchy
  - Distributed name servers
  - Know first one, it knows upper level
  - Local responses cached
    - Local DNS, and at host

`nslookup, dig, host`

Initial latency may be high for first query (then cached by host)

Slide 25

Outline

- Introduction (done)
- Basic Internet Architecture (done)
- Loss and Latency (next)
- Latency Compensation Techniques
- Client-Server Synchronization

Slide 26

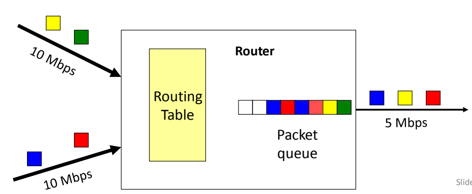
Loss and Latency

- Characteristics most identified with IP networks
  - Note, in other cases capacity, but not usually for network games
- **Loss** – packet does not arrive
  - Usually, fraction `#recv/#sent`, p → `[0:1]`
  - Note, often assumed independent but can be bursty (several lost in row)
- **Latency** – time to get from source to destination
  - Round trip time (RTT) often important since server response to game action
  - Often assumed (2 x latency), but network path can be asymmetric
  - Also jitter (or *latency jitter*) variation in latency (not discussed more here)
- How much does each matter? (later)
- Right now, sources for each

Slide 27

Sources of Loss

- Note, here we are considering only *IP* packet loss
  - Above IP, TCP will retransmit lost packets
  - Below IP, data link often retransmits or does repair (forward error correction)
- IP packet loss predominantly from congestion
  - Causes queue overflow (incoming packets dropped)
- Bit errors
  - More common on wireless
- Loss during route change (link/host unavailable)
- Often bursty!



Slide 28

Sources of Latency

- Serialization – Time to transmit packet on link 1 bit at a time
- Propagation – Time for bits to travel from one host to another
- Queueing delay – Time spent in router queue waiting to be transmitted

- Typically
  - Propagation time fast (about speed of light)
  - Serialization time usually fast for high capacity links
  - Queuing delay can dominate, and exacerbated by long serialization times

Slide 29

Latency Compensation Mini-Outline

- Motivation
- Prediction
- Time delay and Time warp
- Data compression
- Visual tricks
- Cheating

Slide 30

Need for Latency Compensation

- Capacities are growing, but cannot solve all problems
- Still bursty, transient congestion (queues)
- Capacities uneven across all clients
  - And often asymmetric in downlink/uplink.
- Wireless Wide Area Networks (WWANs) growing (low, variable capacities, high latency)
- Propagation delays (~25 msec min across U.S.)

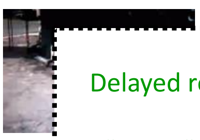
"There is an old network saying: 'Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed – you can't bribe God.' " —David Clark, MIT

Slide 31

Is It Latency or Do You Just Suck?

Slide 32

Is It Latency or Do You Just Suck?

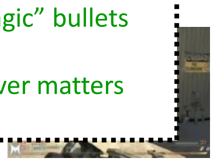


http://ww

Delayed response

"Magic" bullets

Server matters

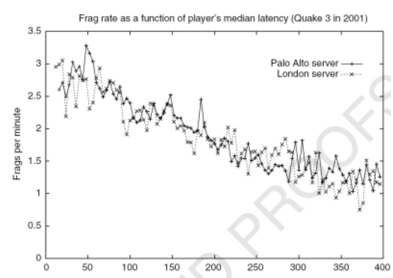


http://www.youtube.com/watch?v=Bn1nBR5jOx8

Slide 33

Latency and Playability (1 of 2)

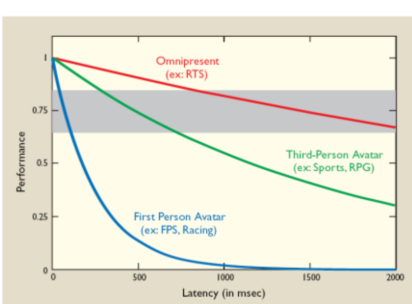
- Affects player – subjective and objective (below)



But depends upon type of game!

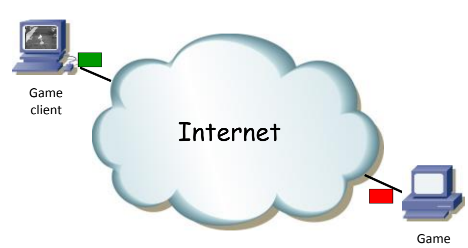
Slide 34

Latency and Playability (2 of 2)



Slide 35

What is Network Latency for Games?



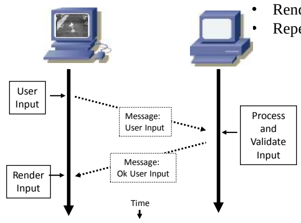
- *Latency* - time to get from source to destination
  - There and back (*round-trip time*)

Slide 36

Basic Client-Server Game Architecture

- "Dumb" client
- Server keeps all state
- Validates all moves
- Client only updates when server says "ok"

Algorithm
- Sample user input
- Pack up data and send to server
- Receive updates from server and unpack
- Determine visible objects and game state
- Render scene
- Repeat



Latency affects *responsiveness*

Slide 37

Latency Example (1 of 2)


Player is pressing left

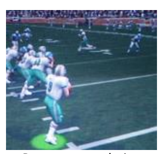

Player is pressing up


Running back goes out of bounds

Slide 38

Latency Example (2 of 2)


Player is pressing "pass"


Pass starts rendering


Interception

Slide 39

Outline

- Introduction (done)
- Basic Internet Architecture (done)
- Loss and Latency (done)
- Latency Compensation Techniques (next)
- Examples – Dragonfly and UE4

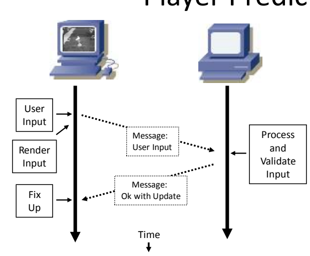
Slide 40

Compensating for Latency – Prediction

- Broadly, two kinds of latency compensation:
  – Player prediction
  – Opponent prediction (often called "dead reckoning" but that name does little to help remember)

Slide 41

Compensating for Latency – Player Prediction



Prediction Algorithm
- Sample user input
- Pack up data and send to server
- Determine visible objects and game state
- Render scene
- Receive updates from server and unpack
- Fix up any discrepancies
- Repeat

Potentially, *tremendous* benefit. Render as if local, no latency. But, note, "fix up" step additional. Needed since server has master copy.

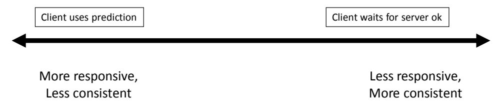
Slide 42

Example of State Inconsistency

- Predicted state differs from actual state

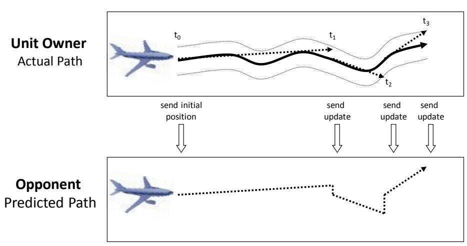
Prediction Tradeoffs

- Tension between responsiveness (latency compensation) and consistency



Compensating for Latency – Opponent Prediction

- Opponent sends position, velocity (maybe acceleration)
- Player predicts where opponent is



(User can see "Warp" or "Rubber band".)

Opponent Prediction Algorithms

Unit Owner

- *Sample* user input
- *Update* {location | velocity | acceleration} on basis of new input
- *Compute* predicted location on the basis of previous {location | velocity | acceleration}
- If (current location – predicted location) < threshold then
  - *Pack up* {location | velocity | acceleration} data
  - *Send* to each other opponent
- Repeat

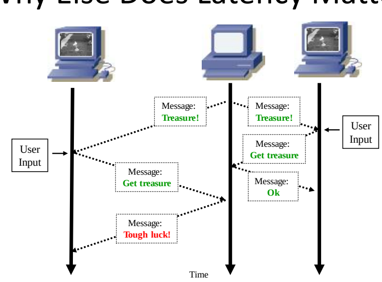
Opponent

- *Receive* new packet
- *Extract* state update information {location | velocity | acceleration}
- If seen unit before then
  - *Update* unit information
- else
  - *Add* unit information to list
- For each unit in list
  - *Update* predicted location
- *Render* frame
- Repeat

Opponent Prediction Notes

- Some predictions easy
  - Ex: falling object
- Other predictions harder
  - Ex: pixie that can teleport
- Some predictions game specific
  - Ex: Can predict "return to base" with pre-defined notion of what "return to base" is.
- Cost is having each host runs prediction algorithm for each opponent.
- Also, although is latency compensation method, can greatly reduce bitrate.
  - Predict self. Don't send updates unless needed.
  - Especially when objects relatively static.

Why Else Does Latency Matter?

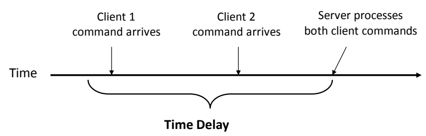


Latency affects *fairness*

Solution? Manipulate time
Time Delay
Time Warp

Compensating for Latency – Time Delay

- Server delays processing of events
  - Wait until all messages from clients arrive
- Server sends messages to more distant client first, delays messages to closer
  - Needs accurate estimate of RTT
- (Note, game plays at highest round trip time (RTT))



Slide 49

Compensating for Latency – Time Warp

- In older FPS (e.g., Quake 3), player had to "lead" opponent to hit
  - Otherwise, opponent had moved
  - Even with "instant" weapon!
- Knowing latency roll-back (warp) to when action taken place
  - Usually assume ½ RTT

Time Warp Algorithm

- Receive packet from client
- Extract information (user input)
- elapsed time = current time – latency to client
- Rollback all events in reverse order to current time – elapsed time
- Execute user command
- Repeat all events in order, updating any clients affected
- Repeat

Slide 50

Time Warp Example

- Client 100 ms behind Server
- Shots still hits (note blood)

Slide 51

Time Warp Notes

- Inconsistency
  - Opponent targets player
  - Player moves around corner to hide
  - Time warps backward → hit
  - Bullets seem to "bend" around corner!
- Fortunately, player often does not notice
  - Doesn't see opponent
  - May be just wounded

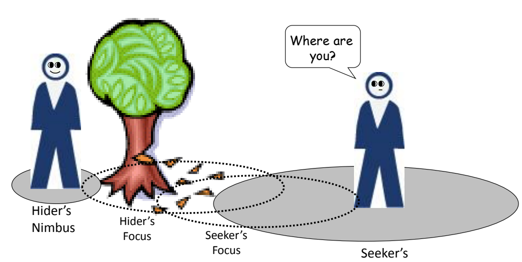
Slide 52

Compensating for Latency – Data Compression (1 of 2)

- Idea → less data, means less latency to get it there
  - So, reduce # or size of messages → reduce latency (serialization)
- Lossless (like zip)
- Opponent prediction
  - Don't send unless need update
- Delta compression (like opponent, but more general)
  - Don't send all data, just updates
- Interest management
  - Only send data to units that need to see it

Slide 53

Interest Management



Slide 54

Compensating for Latency – Data Compression (2 of 2)

- Peer-to-Peer (P2P)
  - Limit server congestion
  - Also, client1→server→client2 higher latency than client1→client2
  - But scales with *slowest* computer
  - But cheating especially problematic in P2P systems
- Update aggregation
  - Message Move A → Send C, Move B → Send C
  - Instead, Move A + Move B → Send C
  - Avoid packet overhead (if less than MTU)
  - Works well w/time delay

Slide 55

Compensating for Latency – Visual Tricks

- Latency present, but hide from user
  - Give feeling of local response
- Ex: player pulls trigger, make sound and puff of smoke while waiting for confirmation of hit
- Ex: player tells boat to move, while waiting for confirmation raise sails, pull anchor
- Ex: player tells tank to move, while waiting, batten hatches, start engine

Slide 56

Outline

- Introduction (done)
- Basic Internet Architecture (done)
- Loss and Latency (done)
- Latency Compensation Techniques (done)
- Client Server Synchronization (next)
  - By Example – Dragonfly and UE4

Slide 57

Network Game Case Study – Saucer Shoot 2

Saucer Shoot for two players

Slide 58

Dragonfly – Network Manager

```
class NetworkManager : public Manager {

 private:
  NetworkManager();.
  NetworkManager(NetworkManager const&);
  void operator=(NetworkManager const&);
  int sock;

 public:

  // Get the one and only instance of the NetworkManager.
  static NetworkManager &getInstance();

  // Start up NetworkManager.
  int startUp();

  // Shut down NetworkManager.
  void shutDown();

  // Accept only network events.
  // Returns false for other engine events.
  bool isValid(string event_type);

  …
```

Basic manager stuff

```
  // Block, waiting to accept network connection.
  int accept(string port = DRAGONFLY_PORT);

  // Make network connection.
  int connect(string host,
              string port = DRAGONFLY_PORT);

  // Close network connection.
  int close();

  // Send buffer to connected network.
  int send(void *buffer, int bytes);

  // Receive from connected network (no more than bytes).
  int receive(void *buffer, int bytes);

  // Check if network data.
  int isData();

  // Return true if network connected, else false.
  bool isConnected();

  // Return socket.
  int getSocket();
};
```

Network specific stuff

Slide 59

Dragonfly – Network Events

```
#include "Event.h"

#define NETWORK_EVENT "__network__"

class EventNetwork : public Event {

 private:
  int bytes;   // Number of bytes available

 public:
  // Default constructor.
  EventNetwork();

  // Create object with initial bytes.
  EventNetwork(int initial_bytes);

  // Set number of bytes available.
  void setBytes(int new_bytes);

  // Get number of bytes available.
  int getBytes();
};
```

- Indicate network data is available
- And how much
- Bytes are actually still with network manager

Slide 60

Client and Host Objects

- Host object (derived from Object) runs on server
- Client object (derived from Object) runs on client
- Host game started first, whereupon Host (using NetworkManager) readies computer for connection
- Client (also using NetworkManager) starts after and connects to Host
- Client gathers input normally, but also sends data to Host
- Host receives keystrokes sent by Client, generating network events to game objects (e.g., the Client Hero) to handle
- Each game loop, Host checks all game objects to see which ones are new and/or updated
  - Need to synchronize Objects between Host and Client … but how?

Slide 61

How to Synchronize Client and Host (Server)?

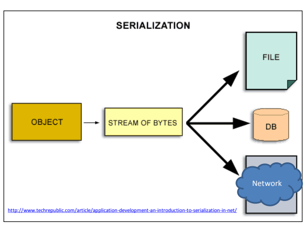
- Many decisions for multiplayer game
  - How are player actions transmitted to server?
  - What Objects are synchronized and how often?
  - How are inconsistencies between client and server game states resolved?
- Key aspect – how to “send” Object from one computer to another

Slide 62

Serializing Objects

- Convert Object attributes to byte stream for storage or transmission

Also known as “marshalling”



Slide 63

Serializing (Marshalling) Objects

Object class extensions to support marshalling

```
// Serialize Object attributes to single string (json-like).
// e.g., "id:110,is_active:true, ...
// Only modified attributes are serialized (unless all is true).
virtual string serialize(bool all = false);

// Deserialize string to become Object attributes.
virtual int deserialize(string s);

// Return true if attribute modified since last serialize.
bool isModified(enum ObjectAttribute attribute);
```

e.g.,

```
id:0,is_active:true,is_visible:true,event_count:0,box-corner-x:0,
box-corner-y:0,box-horizontal:1,box-vertical:1,pos-x:0,pos-y:0,
type:Object,sprite_name:,sprite_center:true,sprite_transparency:0,
sprite_index:0,sprite_slowdown:1,sprite_slowdown_count:0,altitude:2,
solidness:0,no_soft:false,x_velocity:0,x_velocity_countdown:0,
y_velocity:0, y_velocity_countdown:0,
```

Slide 64

Synchronizing Objects (1 of 2)

- Only synchronize important objects and events (e.g., Hero destruction vs. Stars moving)

Synchronize	Don't Synchronize
Saucer creation/destruction	Stars
Bullet creation/destruction	Object movement that velocity handles
Hero creation/destruction	Explosions
Points increase	Reticle
Above Object position changes	

Synchronizing Objects (2 of 2)

- Have configuration for game (host | client)
  - Can keep same codebase for Hero, Bullet, Points…
- Generally, only Host creates/destroys
  - Except for Explosion
- Generally, Host and Client move and animate
  - Except for Client Hero (see below)
- Client Player input
  - Could update Hero location and synchronize
    - But if not allowed, need to “roll back” state
  - Instead, send keystrokes to server
    - Let server move all Objects and send to client

Slide 66

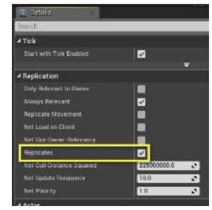
Multi-player Networking

- Client-Server
  - Authoritative server, makes all decisions
- Replicate objects, variables, functions
  - Replicated Actors are main “workhorse” server uses to synchronize
  - Server gathers attributes that change, send to client
- Not all objects, variables, functions need to be replicated
  - E.g., objects that compute AI behavior on server → only when move Actor
  - E.g., only replicate functions that result in client seeing/hearing

Slide 67

Replication

- When replicated object created/modified/destroyed on server, sent to clients
- When replicated object created/modified on client, not sent
  - Can be used for “cosmetic” objects that don’t effect gameplay
- Client sends information via “run on server” functionality



Slide 68

Remote Procedure Calls (RPC)

- RPCs are functions called locally (they look like “normal” functions), but are executed on server
- Allow client/server to send messages to each other
- Used for playing sounds, spawning particles
- Also, client invokes via “run on server” function

Slide 69

Reliability

- Any replicated event can be reliable or unreliable
- Reliable
  - Guaranteed to be called, resent if error, delayed when bandwidth saturated
    - E.g., use for starting game
- Unreliable
  - Attempt to call, but not resent if error, dropped if bandwidth saturated
    - E.g., use for player movement
- NetMulticast – send to all clients (not true multicast)
  - E.g., send notice of player death

Slide 70

Summary

- Networking increasingly important for games
  - The network *is* the computer
  - Many games come with multi-player, online play, downloads, player communities
- Internet influences design of game architecture
  - Need to live with “best effort” service
- Choice of solution depends upon action within game
  - Transport protocol
  - Latency compensation
  - Client-server architectures dominate
- Game developers need to carefully consider design of object synchronization

Slide 71