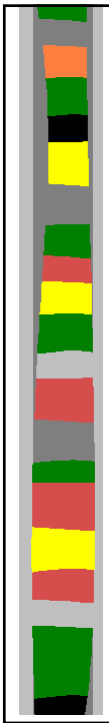
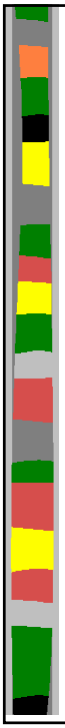


Networking for Games

IMGD 4000

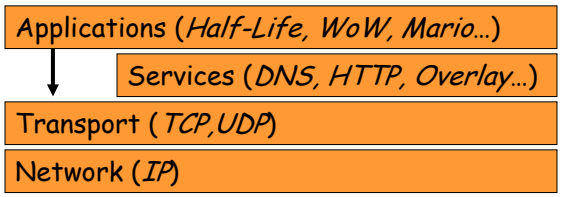


Outline

- Introduction
- Basic Internet Architecture
- Loss, Latency and Jitter
- Latency Compensation Techniques
- Playability versus Network Conditions

Introduction

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



The Internet from the Edge

- Reasonable analogy → *Postal Service*
  - Letters in envelopes
  - Address envelopes
  - Put in Mailbox → trust that reach destination
  - Don't know how they get there
  - Delivery takes different amounts of time
    - Generally, further away longer (but not always)
  - Use external ways to confirm
    - (ex: Use phone, or resend letter until confirmation)
- Users view as an opaque cloud
  - An Internet packet is a like a *letter*
  - The IP address is like the address on an envelope

Provides "Best Effort" Service

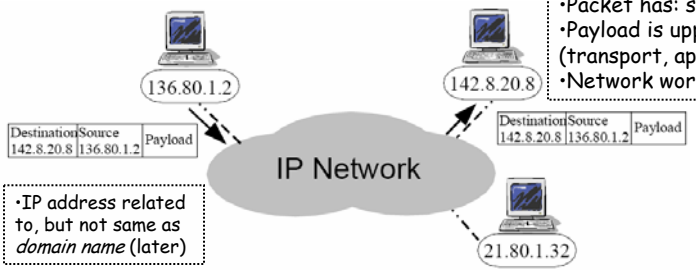
- Few guarantees on timeliness
  - Take milliseconds, 100's of milliseconds, or even seconds
- Few guarantees on arrival certainty
  - Sometimes a packet doesn't arrive (*loss*)
  - Or can arrive twice
  - Or arrives out of order
- Time to reach destination called *latency*
  - *Lag* typically latency + end-host (server an client) time
    - Often, users have a hard time distinguishing
- Short-term variation in latency called *jitter*

(More on *loss*, *latency* and *jitter* Chapter 5)

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} addresses, but practically fewer since allocated in blocks
- Each Internet host has IP address
  - Client running game client
  - Server running game server
- Some have 2
  - Client with wireless and wired network (multi-homed)
  - Router with multiple connections

IPv6 has 2^{128} addresses, but not widely deployed for next 10 years



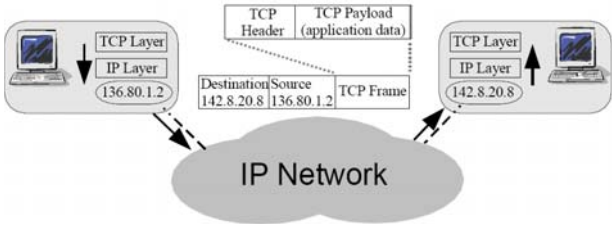
- Packet has: source, destination
- Payload is upper layer (transport, application)
- Network worries about arrival

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

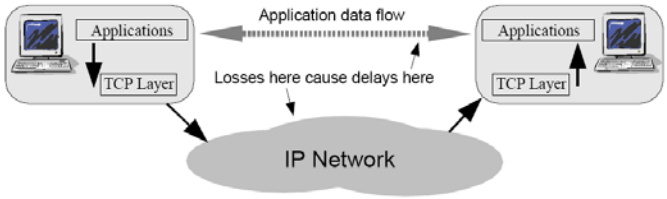
Transmission Control Protocol

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

Many games more sensitive to time!
→Don't use TCP
But many do!
→RTS, MMO



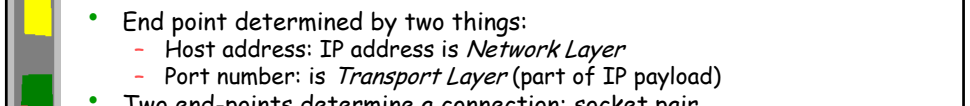
User Datagram Protocol



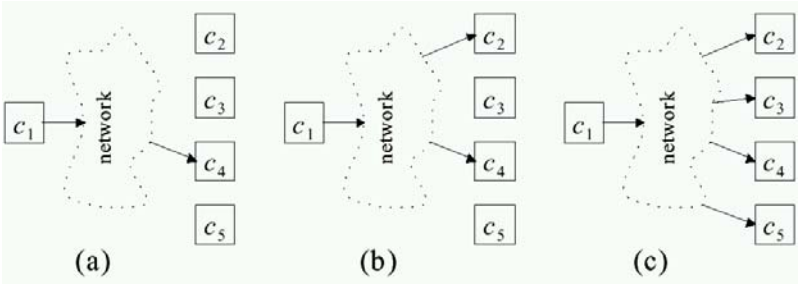
- Some applications sensitive to time
  - Ex: Voice over IP (VoIP)
  - Some games (First-Person Shooter)
- Unreliable
- Connectionless
- No flow control (sender goes faster than receiver)
- No congestion control (sender goes faster than network)
  - Note: IP does ensure there are no bit errors (via Cyclic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

Multiplexing and Flows

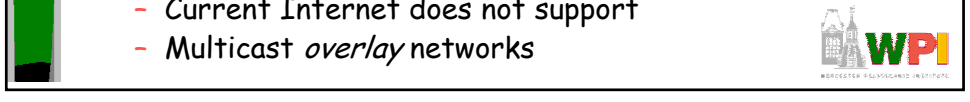
- End point determined by two things:
  - Host address: IP address is *Network Layer*
  - Port number: is *Transport Layer* (part of IP payload)
- Two end-points determine a connection: socket pair
  - ex: `206.62.226.35,p21` + `198.69.10.2,p1500`
  - ex: `206.62.226.35,p21` + `198.69.10.2,p1499`
- Numbers (typical, since vary by OS):
  - `0-1023` "reserved", must be root
  - `1024 - 5000` "ephemeral"
  - Above `5000` for general use
- Well-known, reserved services (see `/etc/services` in Unix):
  - `ftp` `21/tcp`
  - `telnet` `23/tcp`
  - `http` `80/tcp`
  - `Quake3` `27960/udp`
  - `Half-Life2` `27016/udp`



Unicast, Multicast, Broadcast

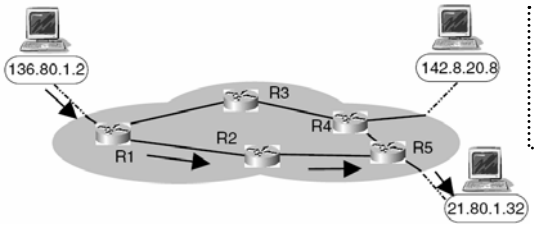


- (a) Unicast, one send and one get
  - Wastes bandwidth when path shared
- (c) Broadcast, one send and all get
  - Perhaps ok for LAN
  - Wastes bandwidth when most don't need
- (b) Multicast, one send and only subscribed get
  - Current Internet does not support
  - Multicast *overlay* networks



Connectivity and Routing

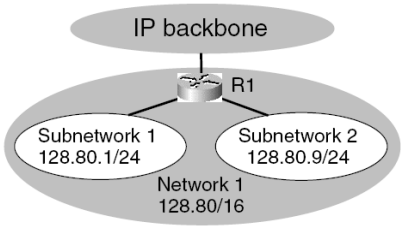
- Often edge most important
  - Game developer does not see internals
- But some aspects critical for understanding network performance



•(Label links, routers)
•Independent choice for packet based *solely* on destination address

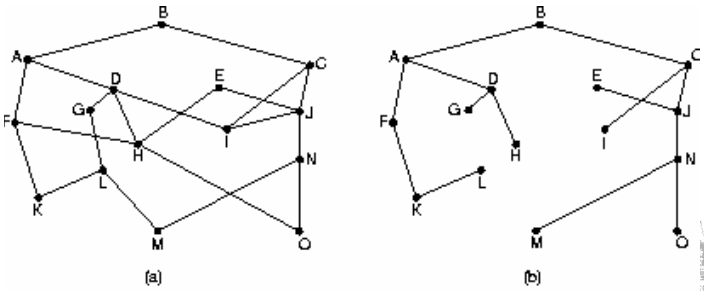
Hierarchy and Aggregation

- 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…



Routing

- Routers use dynamic
  - Discover topology
  - Pick "best" routes (want tree)
    - Typically shortest path (# hops, latency…)
- Note: Local (internal to ISP) routing protocol different than among ISPs (ASes)
  - "Cost" between ASes different

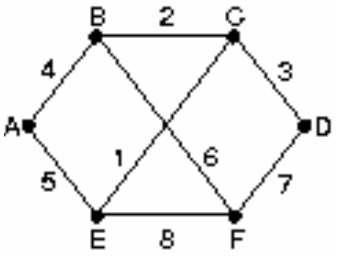


Link State Routing

- Used (w/variations) on Internet since 1979
  - Open Shortest Path First (OSPF)
- Basic steps
  - Discover neighbors (upon boot)
  - Experimentally measure distance (ping/echo)
  - Construct a packet telling what learned
    - (next slide)
  - Send to all other routers
  - Compute shortest path
    - (slide after that)

Constructing Link State Packets

- Identity of sender, sequence number, age, list of (neighbors + distance)



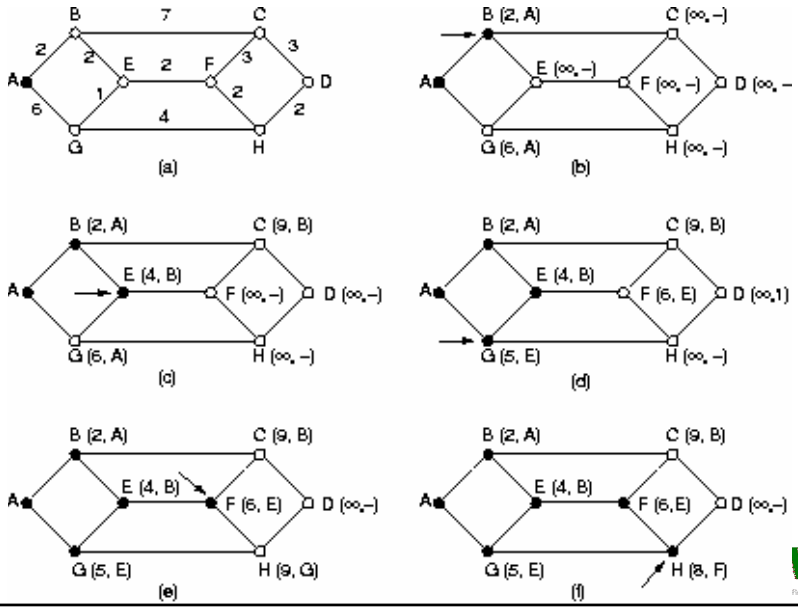
Link State

A	B	C	D
Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age
B 4	A 4	B 2	C 3
E 5	C 2	D 3	F 7
	F 6	E 1	

Computing the Shortest Path

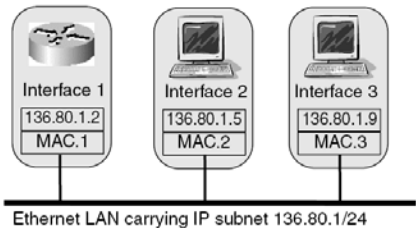
- Dijkstra's Algorithm (1959)
  - Greedy algorithm (add next shortest)
  - $O(|V|^2+|E|)$ (V vertices, E edges)
- Label each node with distance from source
  - if unknown, then ∞
- As algorithm proceeds, labels change
  - tentative at first
  - permanent when "added" to tree
- Note, done on each node

Dijkstra's Algorithm: A to D



Link Layer

- Map IP address to data link layer
  - Medium Access Control (MAC)
  - Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11)
  - MAC address specified by vendor on card
  - 48-bit: `00:0F:1F:81:41:6C`
- Assignment:
  - Fixed (register with netops)
  - Dynamic (assigned when boot)



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

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

- Maximum Transmission Unit (MTU)
  - IP packet could be 64 kbytes
  - In practice, bound by Ethernet (prevalent standard)

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

Address Management Mini-Outline

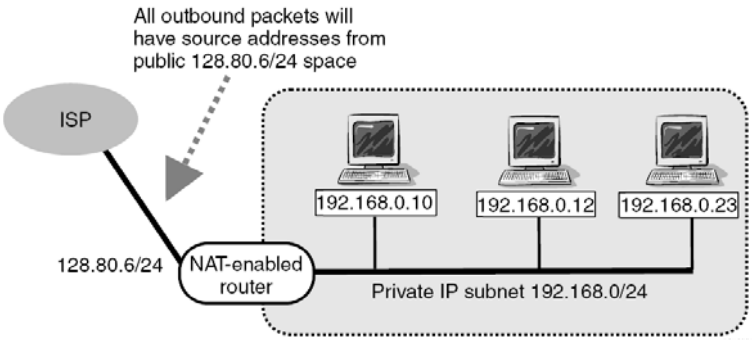
- Network Address Translation
- Dynamic Host Configuration Protocol
- Dynamic Name Service

Network Address Translation (NAT) (1 of 2)

- Used at boundary of ISP
  - Where internal address on publicly routable external 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)

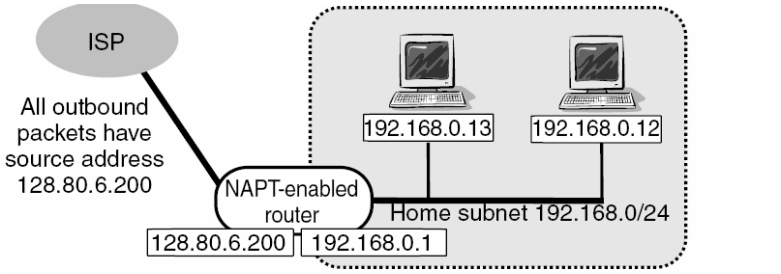
Network Address Translation (NAT) (2 of 2)

- Source hosts use private IP
- Forward to NAT router
- Swap source address with public address (could be range)
- Send to ISP
- Remember process so can do reverse on return



Network Address Port Translation (1 of 2)

- Have only 1 public IP for multiple private IP computers



Addresses and ports are re-mapped on the way to ISP

Source 192.168.0.12:W becomes Source 128.80.6.200:Y

Source 192.168.0.12:X becomes Source 128.80.6.200:Z

Network Address Port Translation (2 of 2)

- Easy to renumber (one number)
- Only need one IP
- Breaks transparency (need to add functionality for each new protocol)
- Hard for outside hosts to access inside
  - Ex: what if two different Quake3 servers inside?
  - Need non-standard ports that *clients* know about
    - Typically, local server register w/master server
      - Gives IP + Port where server is
  - Need to configure NAT box to forward ports

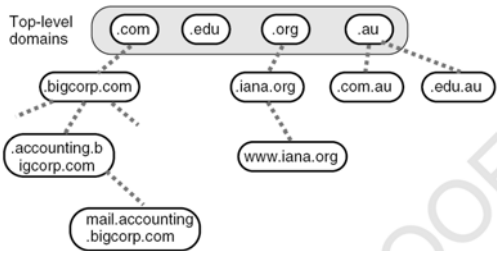
Dynamic Host Configuration Protocol (DHCP)

- Hosts need: IP address, subnet mask, IP of at least one router
  - Use DHCP to get from a LAN device
    - Typical with WLAN router, cable modem, …
- Client broadcasts DHCP discovery to port 67
  - Identifies its MAC
- DHCP server responds w/IP + Mask + Router IP
- Client confirms, selects from server (could be more than one DHCP server)
- Server ACKs

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`

`nslookup, dig, host`



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

Outline

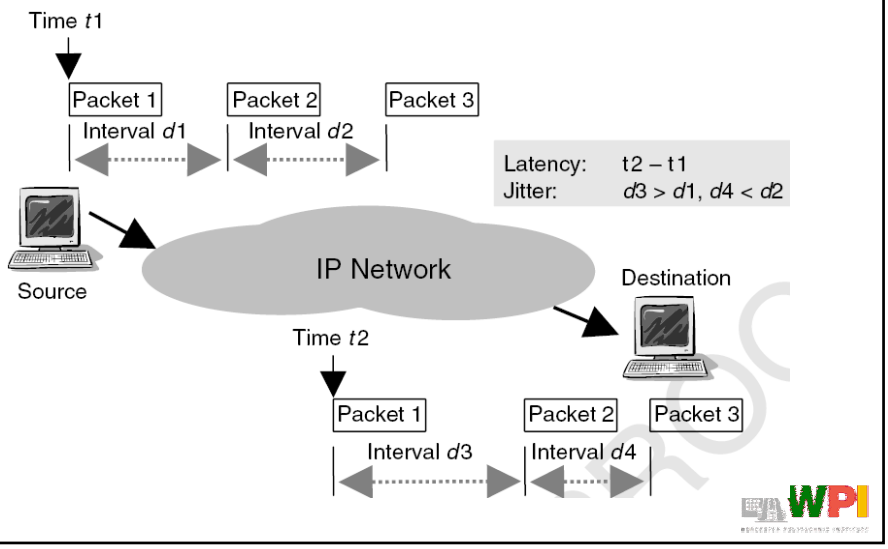
- Introduction (done)
- Basic Internet Architecture (done)
- Loss, Latency and Jitter (next)
- Latency Compensation Techniques
- Playability versus Network Conditions

Latency, Jitter and Loss

(See Picture next slide)

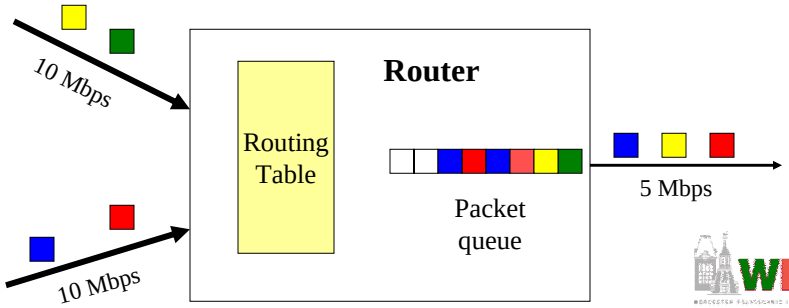
- 3 characteristics most identified with IP networks
  - Note: bandwidth? Sometimes. (More later)
- *Loss* - packet does not arrive
  - Usually, fraction `#recv/#sent`, `p→[0:1]`
  - Note, often assumed independent but can be bursty (several lost in a row)
- *Latency* - time to get from source to destination
  - Round trip time (RTT) often assumed to be 2*latency, but network path can be asymmetric
- *Jitter* - variation in latency
- How much does each matter? (Chapter 7, later)
- Right now, sources for each

Latency, Jitter and Loss



Sources of Loss

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



Sources of Latency Mini-Outline

- Propagation
- Serialization
- Congestion

Sources of Latency - Propagation Delay

- Time for bits to travel from one host to another
- Limited by propagation speed of medium
  - Typically electricity/light through cable or fiber
  - Could be radio wave through air
  - Could even be sound wave through water!
- Roughly:

```
latency (ms) = length of link (km) / 300
```

- Ex: Worcester, MA to Berkeley, CA is 2649 miles (4263 km)

```
latency = 4263 / 300 = 14 msec
```

- Notes:
  - Light through fiber about 30% slower than light through vacuum
  - Paths often not in a straight line

Sources of Latency - Serialization Delay

- Ex: Consider everyone trying to leave room by one door
  - Exit only at fixed rate
  - Similar to transmitting bits by a network card
- Time to transmit packet on link 1 bit at a time → *serialization*
- Serialization delay for each hop (cumulative)
- Includes headers (26 bytes for Ethernet)

```
latency (ms) = 8 * link layer frame (bytes) /
                   link speed (kbps)
```

- Ex: 1000 byte app data, uplink typical DSL rate
  - Frame is 1000 + 40 (UDP/IP) + 26 (Ethernet)

```
latency = 8 * 1066 / 192 = 44 msec
```

Sources of Latency - Queuing Delay

- When traffic rate bursty, unpredictable rate (unlike, say, phone)
  - Need to handle burst → queue
- Queuing delay

```
latency (ms) = 8 * queue length (packets) *
                   avg pckt sz (bytes) / link speed (kpbs)
```

- Ex: 10 packets, each 1000 bytes, 1 Mbps link

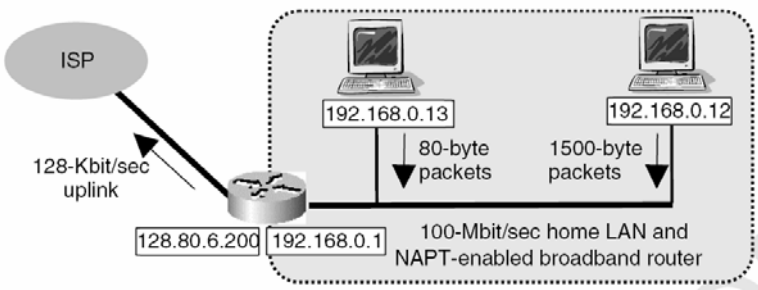
```
latency = 8 * 10 * 1000 / 1000 = 80 msec
```

- Note, can have at end-host, too, when send faster than link (ie- WLAN)

`ping, traceroute (Linux), tracert (Windows)`

Sources of Jitter

- Due to a change in end-to-end delay from one packet to the next
- Route changes
- Queue length changes
  - Say, goes from 10 (80 msec delay) to 0
- Packet length changes (serialization different)
  - Big packet (1000 bytes) → 44 msec
  - Small packet (10 bytes) → 4.4 msec
  - Could be from other packets in the queue, too



Tools

- `ping`
  - http://www-iepm.slac.stanford.edu/pinger/
- `traceroute`
  - http://www.traceroute.org
- bandwidth estimation
  - http://www.speedtest.net/index.php
  - http://speedtest.verizon.net/SpeedTester/help_speedtest.jsp

```
 1  FastEthernet6-0.civ-service1.Canberra.telstra.net
    (203.50.1.65)  0.225 ms  0.193 ms  0.268 ms
          [..]
 7  i-7-0.syd-core02.net.reach.com (202.84.221.90)  5.457 ms
    5.636 ms  5.349 ms
 8  i-0-0.wil-core02.net.reach.com (202.84.144.101)  153.923 ms
    153.935 ms  154.057 ms
          [..]
11  0.so-3-0-0.CL1.LAX15.ALTER.NET (152.63.117.90)  153.84 ms
    154.67 ms  154.303 ms
12  0.so-5-0-0.XL1.NYC9.ALTER.NET (152.63.0.174)  227.725 ms
    265.947 ms  227.653 ms
          [..]
17  192.11.226.2 (192.11.226.2)  228.585 ms  229.29 ms  227.779 ms
```

•Note, ~145 ms (12,000 km Sydney to LA) when estimate is 80 ms

Latency Compensation Mini-Outline

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

Need for Latency Compensation

- Bandwidth is growing, but cannot solve all problems
- Still bursty, transient congestion (queues)
- Bandwidth upgrade uneven across all clients
  - Modems? Maybe. DSL, yes, but even those vary in downlink/uplink.
- WWAN growing (low, variable bandwidth, high latency)
- Propagation delays (~25 msec minimum to cross country)

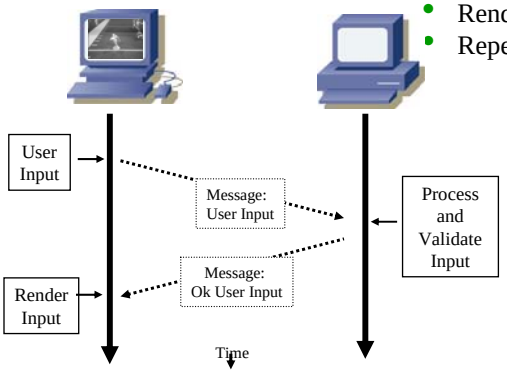
"There is an old network saying: 'Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed – you can't bribe God.' " —David Clark, MIT

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 Example (1 of 2)

Player is pressing left

Player is pressing up

Running back goes out of bounds

Latency Example (2 of 2)

Player is pressing "pass"

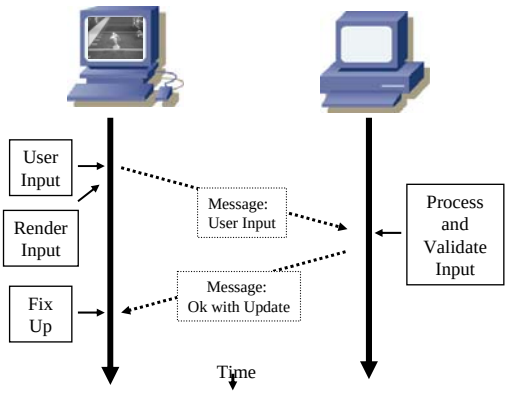
Pass starts rendering

Interception

Compensating for Latency - Prediction

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

Player Prediction



Predicted 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

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

Example of State Inconsistency

- Predicted state differs from actual state

Prediction Tradeoffs

- Tension between responsiveness (latency compensation) and consistency.

Client uses prediction ⟷ Client waits for server ok

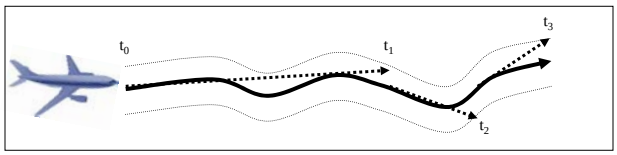
More responsive,
Less consistent

Less responsive,
More consistent

Opponent Prediction

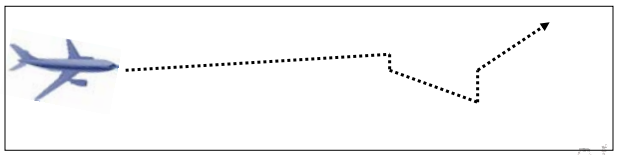
- Opponent sends position, velocity (maybe acceleration)
- Player predicts where opponent is

Unit Owner
Actual Path



send initial position | send update | send update | send update

Opponent
Predicted Path



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

Opponent Prediction Algorithms

Unit Owner

- *Sample* user input
- *Update* {`location` | `velocity` | `acceleration`} on the 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

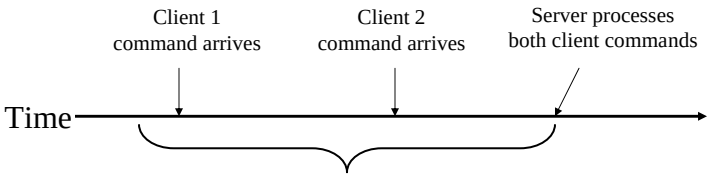
- Some predictions easy
  - Ex: falling object
- Others harder
  - Ex: pixie that can teleport
- Can be game specific
  - Ex: Can predict "return to base" with pre-defined notion of what "return to base" is.
- Cost is each host runs prediction algorithm for each opponent.
- Also, although a latency compensation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

Time Manipulation

- Client states can differ
  - Depends upon their RTT to server
- Impacts fairness
  - Ex: Two players defeat monster
  - Server generates treasure. Sends messages to clients.
  - Clients get messages. Players can react.
  - Client closer (RTT lower) gets to react sooner, gets treasure
    - Unfair!
- Solution? Manipulate time
  - *Time Delay*
  - *Time Warp*

Time Delay

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



Time Warp

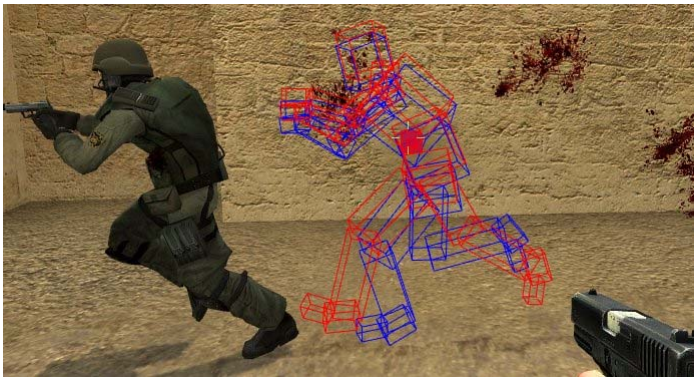
- In older FPS (ie- Quake 3), used to have to lead opponent to hit
  - Otherwise, player had moved
  - Even with "instant" weapon!
- Knowing latency roll-back (warp) to when action taken place
  - Usually assume ½ RTT

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

Time Warp Example

- Client 100 ms behind
- Still hits (note the blood)
- Also, note the bounding boxes



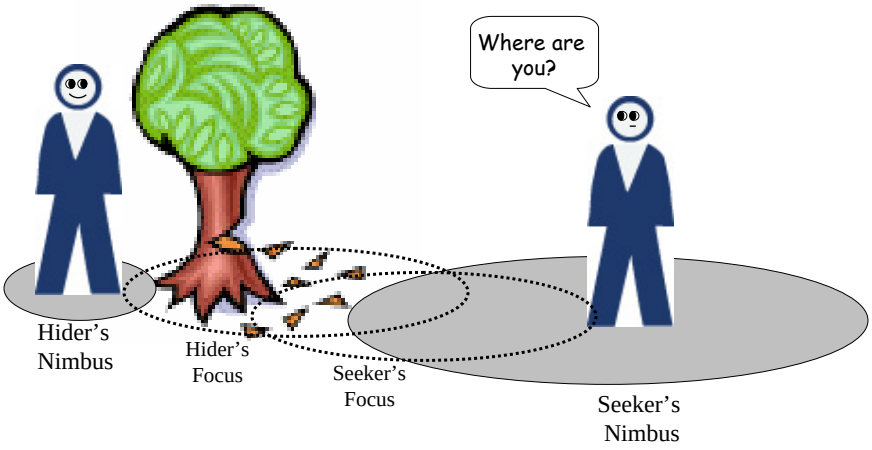
Time Warp Notes

- Inconsistency
  - Player target
  - Move around corner
  - Warp back → hit
  - Bullets seem to "bend" around corner!
- Fortunately, player often does not notice
  - Doesn't see opponent
  - May be just wounded

Data Compression

- 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

Interest Management



Data Compression (continued)

- Peer-to-Peer (P2P)
  - Limit serer congestion
  - Also, client1→server→client2 higher latency than client1→client2
  - 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

Visual Tricks

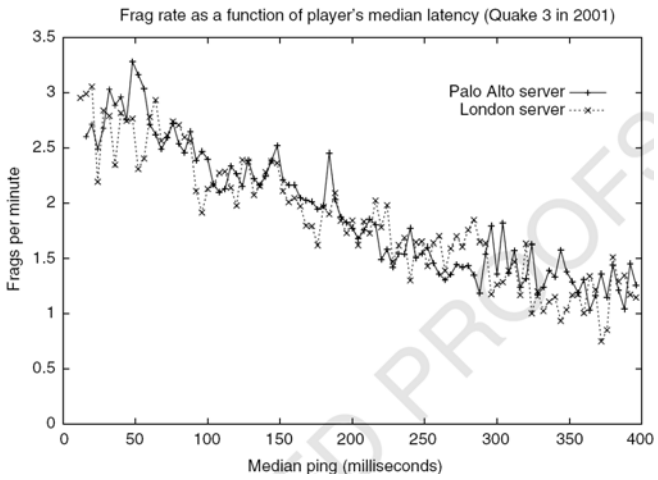
- Latency present, but hide from user
  - Give feeling of local response
- Ex: player tells boat to move, while waiting for confirmation raise sails, pull anchor
- Ex: player tells tank to move, while waiting, batten hatches, start engine
- Ex: player pulls trigger, make sound and puff of smoke while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

Latency Compensation and Cheating

- Opponent prediction → no server is needed!
  - Yes, if player can be *trusted*
  - Else "I just shot you in the head" → how to verify?
- Time warp → client pretends to have high latency
  - Can pass to player then react
  - Worse if client controls time stamps
- Interest management can help with information exposure

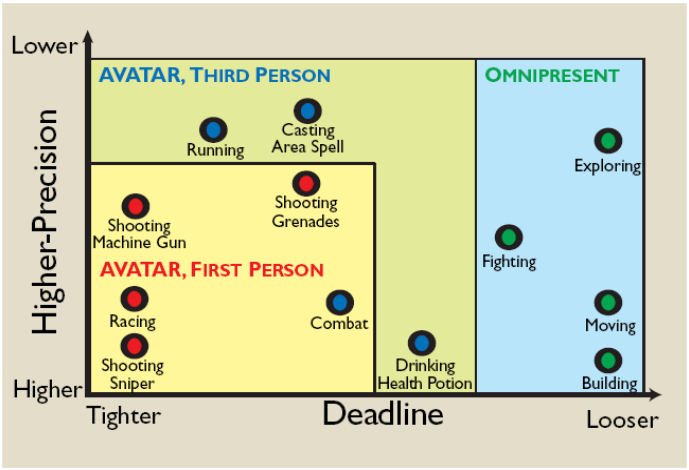
Networking and Playability

- Latency affects performance
  - Subjective and Objective

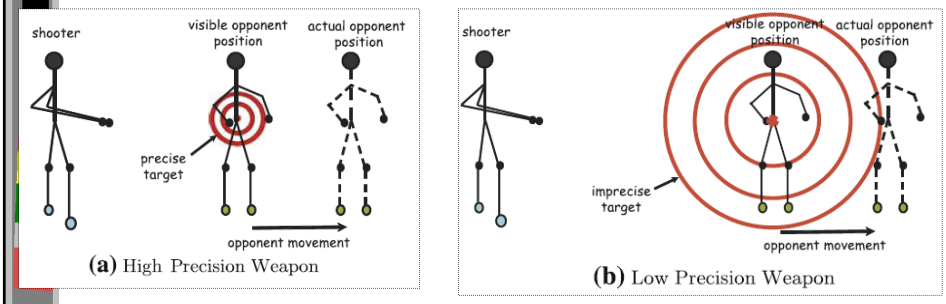


But depends upon task!

Precision and Deadline

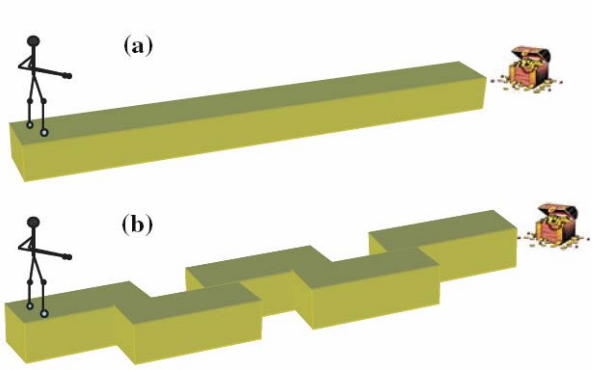


Precision Example



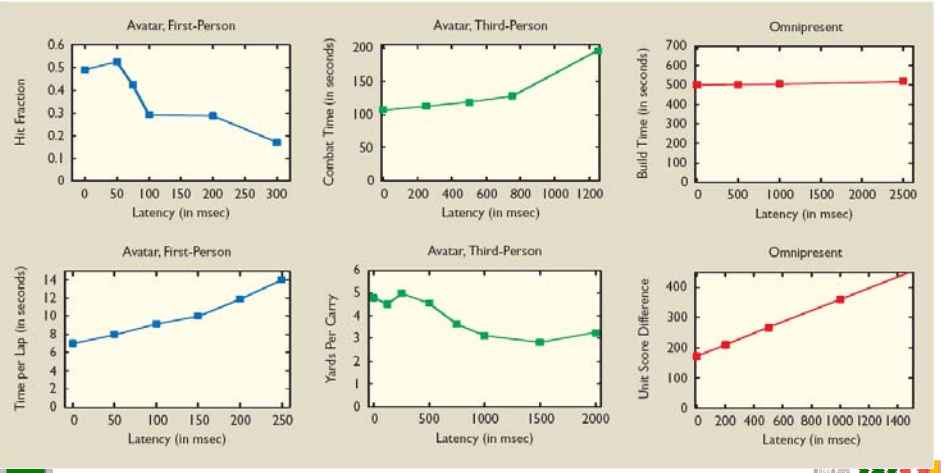
Shooting an opponent in a FPS Game
(a) high precision weapon
(b) low precision weapon

Deadline Example

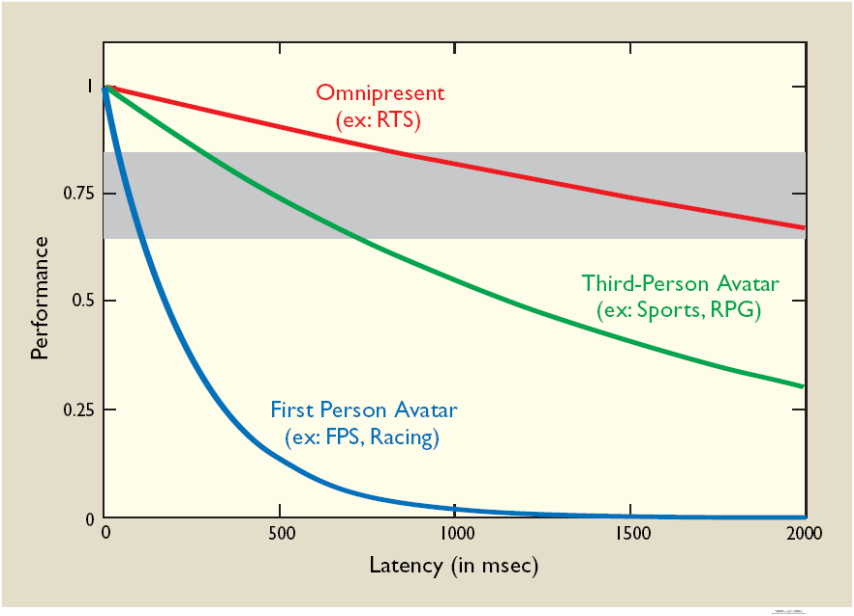


Moving in a FPS Game
(a) Loose deadline
(b) Tight deadline

Player Performance vs. Latency



Player Performance vs. Latency



Networking Cheating in General

- Unique to games
  - Other multi-person applications don't have
  - In DIS, military not public and considered trustworthy
- 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

Preventing Packet 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

Information Exposure

- Allows cheater to gain access to replicated, hidden game data (i.e. status of other players)
  - Passive, since does not alter traffic
  - Example: defeat "fog of war" in RTS, see through walls in FPS
- Cannot be defeated by network alone
- Instead:
  - Sensitive data should be encoded
  - Kept in hard-to-detect memory location
  - Centralized server may detect cheating (example: attack enemy could not have seen)
    - Harder in replicated system, but can still share

Design Defects

- If clients trust each other, then if client is replaced and exaggerates cheater effects, others will go along
  - Can have checksums on client binaries
  - Better to have trusted server that puts into play client actions (centralized server)
- Distribution may be the source of unexpected behavior
  - Features only evident upon high load (say, latency compensation technique)
  - Example: Madden Football

Summary

- Networking increasingly important for games
  - The network *is* the computer
  - Many games come with online play, downloads, player communities
- Internet influences design of game architecture
  - Need to live with "best effort" service
- Choice of solution (latency compensation or transport protocol) depends upon action within game