CS 378: Computer Game Technology

Networking Basics for Games

Spring 2012

Networking for Games

- You need networking for multi-player gaming
- There are *persistent* games, like EverQuest, where state remains regardless whether or not anyone is playing
- There are transient games that exist only while people are playing, and reset each time the server-side is reset
- There are four primary concerns in building networks for games:
  - **Latency**: How long does it take for state to be transmitted
  - **Reliability**: How often is data lost or corrupted
  - **Bandwidth**: How much data can be transmitted in a given time
  - **Security**: How is the game-play protected from tampering
- All of these considerations interact, and trade-offs must be made

Latency in Games

- Recall that latency is the time between when the user acts and when they see the result
- Latency is arguably the most important aspect of a game network
  - Too much latency makes the game-play harder to understand because the player cannot associate cause and effect
  - It makes it harder to target objects (the lead becomes to large)
  - There is significant psychological research on this topic
- Latency is **not** the same as bandwidth
  - A freeway has higher bandwidth than a country road, but the speed limit, and hence the latency, can be the same
  - Excess bandwidth can reduce the *variance* in latency, but cannot reduce the minimum latency (queuing theory)

Sources of Latency

- Consider a client sending and receiving data from a server
- There are four sources of latency in a game network
  - Frame rate latency: Data only goes out on or comes in from the network layer once per frame, and user interaction is only sampled once per frame
  - Network protocol latency: It takes time for the operating system to put data onto the physical network, and time to get it off a physical network and to an application
  - Transmission latency: It takes time for data to be transmitted to the receiver
  - Processing latency: The time taken for the server (or client) to compute a response to the input
- You cannot make any of these sources go away
  - You don't even have control over some of them
  - Remember Amdahl's law when trying to improve latency

Reducing Latency (1)

- Frame rate latency:
  - Increase the frame rate (faster graphics, faster AI, faster physics)
- Network protocol latency:
  - Send less stuff (less stuff to copy and shift around)
  - Switch to a protocol with lower latency
  - But may have impact on reliability and security
- Transmission latency:
  - Send less stuff – less time between when the first bit and the last bit arrive
  - Upgrade your physical network

Reducing Latency (2)

- Processing latency:
  - Make your server faster
  - Have more servers
- The sad fact is, networking researchers and practitioners are almost never concerned with latency
  - Most applications can handle higher latency (who else cares about latency?)
  - When did you last hear a DSL/Cable add that promised lower latency?

Working With Latency

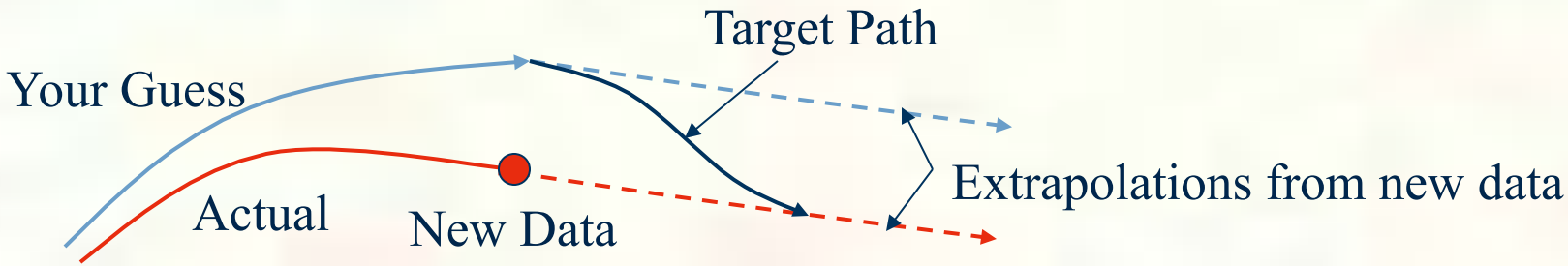
- If you can't get rid of latency, you can try to hide it
- Any technique will introduce errors in some form - you **cannot** provide immediate, accurate information
- Option 1: Sacrifice accurate information, and show approximate positions
  - Ignore the lag and show a given player "old" information about the other players
  - Try to improve upon this by guessing where the other players are. But if your guess is wrong, incorrect information is shown
- Option 2: Sacrifice game-play:
  - Deliberately introduce lag into the local player's experience, so that you have enough time to deal with the network

Dead Reckoning

- *Dead reckoning* uses prediction to move objects about even when their positions are not precisely known, reducing the *appearance* of lag
  - Each client maintains precise state for some objects (e.g. local player)
  - Each client receives *periodic* updates of the position of everyone else, along with velocity information, and maybe acceleration
  - On each frame, the non-local objects are updated by extrapolating their most recent position using the available information
- With a client-server model, each player runs their own version of the game, while the server maintains absolute authority

Fixing Extrapolation Errors

- What do you do when using dead reckoning, and a new position arrives for another player?
  - The position that just came in will not agree with the place you have the object, due to extrapolation errors
- Two options:
  - Jump to the correct position
  - Interpolate the two positions over some period
    - Path followed will never be exact, but will match reasonably well



Network Reliability

- Some protocols attempt to ensure that every packet is delivered
  - It costs, in latency and bandwidth, to ensure delivery
- Others try less hard to ensure delivery, and will not tell you if packets get lost
  - Latency and bandwidth requirements are lower for such protocols
- Other aspects of reliability are error checking (do the right bits arrive?) and order consistency (do things arrive in the same order they were sent?)
- In a game, does everything need to be completely reliable?
- Are all aspects of reliability equally important?

Reliability Requirements

- Some information must be communicated:
  - Discrete changes in game state - if they go missing, there is no chance to recapture them
  - Information about payments, joining, dropping, …
- Some information does not need to be reliably communicated:
  - Information that rapidly becomes out of date, and hence is sent frequently
  - Player position information, weapon firing information, …
- The data that goes out of date quickly is also sent more often; big payoffs for reducing the cost of sending it

Internet Protocols

- There are only two internet protocols that are widely deployed and useful for games: *UDP* and *TCP/IP*
  - TCP/IP (Transmission Control Protocol/Internet Protocol) is most commonly used
  - UDP (User Datagram Protocol) is also widely deployed and used
- Other protocols exist:
  - Proprietary standards
  - *Broadcast* and *Multicast* are standard protocols with some useful properties, but they are not widely deployed
  - If the ISPs don't provide it, you can't use it

TCP/IP Overview

- Advantages:
  - Guaranteed packet delivery
  - Ordered packet delivery
  - Packet checksum checking (some error checking)
  - Transmission flow control
- Disadvantages:
  - Point-to-point transport
  - Bandwidth and latency overhead
  - Packets may be delayed to preserve order
- Uses:
  - Data that must be reliably sent, or requires one of the other properties
  - Games that can tolerate latency

UDP Overview

- Advantages:
  - Packet based - so works with the internet
  - Low overhead in bandwidth and latency
  - Immediate delivery - as soon as it arrives it goes to the client
- Disadvantages:
  - Point to point connectivity
  - No reliability guarantees
  - No ordering guarantees
  - Packets can be corrupted
  - Can cause problems with some firewalls
- Uses:
  - Data that is sent frequently and goes out of date quickly

Choosing a Protocol

- The best way to do it is decide on the requirements and find the protocol to match
- You can also design your own "protocol" by designing the contents of packets
  - Add cheat detection or error correction, for instance
  - You then wrap you protocol inside TCP/IP or UDP

Reducing Bandwidth Demands

- Bandwidth is plentiful on the internet today, so it only becomes an issue with large environments
  - Even "slow" modems have more impact through high latency than low bandwidth (due to compression, error checking and analogue/ digital conversion)
- Regardless, smaller packets reduce both bandwidth *and* latency
  - Latency is measured from the time the first bit leaves to the time the last bit arrives - so fewer bits have lower latency
- There are two primary ways to reduce bandwidth demands:
  - Dead reckoning allows you to send state less frequently
  - *Area of interest management* avoids sending irrelevant data

Area of Interest Management

- *Area of interest management* is the networking equivalent of visibility - only send data to the people who need it
- There is a catch, however: In a network you may not know where everyone is, so you don't know what they can see
  - A chicken-and-egg problem
- Hence, area-of-interest schemes are typically employed in client-server environments:
  - The server has complete information
  - It decides who needs to receive what information, and only sends information to those who need it
- Two approaches: *grid methods* and *aura methods*
  - Sound familiar? (replace aura with bounding box)

Grid and Aura Methods

- Grid methods break the world into a grid
  - Associate information with cells
  - Associate players with cells
  - Only send information to players in the same, or neighboring, cells
  - This has all the same issues as grid based visibility and collision detection
- Aura methods associate an aura with each piece of information
  - Only send information to players that intersect the aura
  - Just like broad-phase collision detection with bounding volumes
- Players need to find out all the information about a space when they enter it, regardless how long ago that information last changed