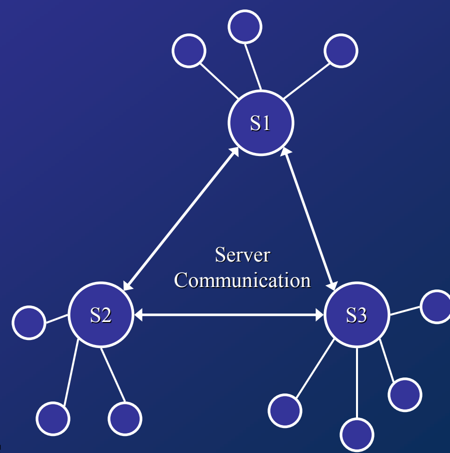
BrickNet (cont'd)

- Object-request brokers on the servers
- Aimed for collaborative design environments
  - each node is responsible for its part of design and for sharing that information
- Also, networked games, groupware systems, concurrent engineering systems, and other asynchronous, network-based graphics environments



Other Academic Projects

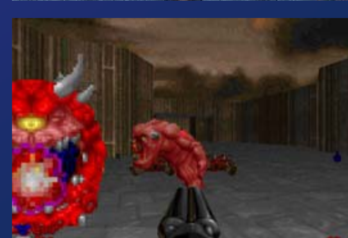
- MASSIVE
  - different interaction media: graphics, audio and text
  - awareness-based filtering: each entity expresses a focus and nimbus for each medium
- Distributed Worlds Transfer and Communication Protocol (DWTP)
  - each object can specify whether a particular event requires a reliable distribution and what is the event's maximum update frequency
- Real-Time Transport Protocol (RTP/I)
  - ensures that all application instances look as if all operations have been executed in the same order
- Synchronous Collaboration Transport Protocol (SCTP)
  - collaboration on closely coupled, highly synchronized tasks
  - the interaction stream has critical messages (especially the last one) which are sent reliably, while the rest are sent by best effort transport

Networked Demos and Games

- SGI *Flight*
  - 3D aeroplane simulator demo for Silicon Graphics workstation, 1983–84
    - serial cable between two workstations
    - Ethernet network
    - users could see each other's planes, but no interaction
- SGI *Dogfight*
  - modification of *Flight*, 1985
  - interaction by shooting
  - packets were transmitted at frame rate → clogged the network
  - limited up to ten players

Networked Games: *Doom*

- id Software, 1993
- First-person shooter (FPS) for PCs
- Part of the game was released as shareware in 1993
  - extremely popular
  - created a gamut of variants
- Flooded LANs with packets at frame rate



Networked Games: 'First Generation'

- Peer-to-peer architectures
  - each participating computer is an equal to every other
  - inputs and outputs are synchronized
  - each computer executes the same code on the same set of data
- Advantages:
  - determinism ensures that each player has the same virtual environment
  - relatively simple to implement
- Problems:
  - persistency: players cannot join and leave the game at will
  - scalability: network traffic explodes with more players
  - reliability: coping with communication failures
  - security: too easy to cheat

1

Networked Games: 'Second Generation'

- Client-server architectures
  - one computer (a server) keeps the game state and makes decisions on updates
  - clients convey players' input and display the appropriate output but do not inlude (much) game logic
- Advantages:
  - generates less network traffic
  - supports more players
  - allows persistent virtual worlds
- Problems:
  - responsiveness: what if the connection to the server is slow or the server gets overburdened?
  - security: server authority abuse, client authority abuse

2

Networked Games: 'Third Generation'

- Client-server architecture with prediction algorithms
  - clients use dead reckoning
- Advantages:
  - reduces the network traffic further
  - copes with higher latencies and packet delivery failures
- Problems:
  - consistency: if there is no unequivocal game state, how to solve conflicts as they arise?
  - security: packet interception, look-ahead cheating

3

Networked Games: 'Fourth Generation'

- Generalized client-server architecture
  - the game state is stored in a server
  - clients maintain a subset of the game state locally to reduce communication
- Advantages:
  - traffic between the server and the clients is reduced
  - clients can response more promptly
- Problems:
  - boundaries: what data is kept locally in the client?
  - updating: does the subset of game state change over time?
  - consistency: how to solve conflicts as they occur?

4

Networked Games: *ARQuake*

- School of Computer and Information Science, University of South Australia
- augmented reality version of *Quake*: walk around in the real world and play *Quake* against virtual monsters
- components
  - head mounted display
  - mobile computer
  - head tracker
  - GPS system

Massive Multiplayer Online Games

Name	Publisher	Released	Subscribers
Ultima Online	Origin Systems	1997	250,000
EverQuest	Sony Entertainment	1999	430,000
Asheron's Call	Microsoft	1999	N/A
Dark Age of Camelot	Sierra Studios	2001	250,000
Sims Online	Electronic Arts	2002	97,000
Star Wars Galaxies	LucasArts	2003	N/A

source: `http://www.mmorpg.com`

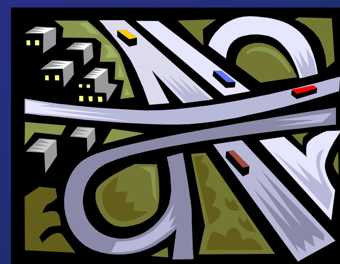
§3 Networking

- Data transfer
  - latency
  - bandwidth
  - reliability
  - protocol
- Internet protocols
  - TCP, UDP
  - unicast, broadcast, multicast
- Communication architectures
  - peer-to-peer
  - client-server

Fundamentals of Data Transfer 1 (3)

- Network latency
  - network delay
  - the amount of time required to transfer a bit of data from one point to another
  - one of the biggest challenges:
    - impacts directly the realism of the NVE experience
    - we cannot much to reduce it
  - origins
    - speed-of-light delay
    - endpoint computers, network hardware, operating systems
    - the network itself, routers

Fundamentals of Data Transfer 2 (3)

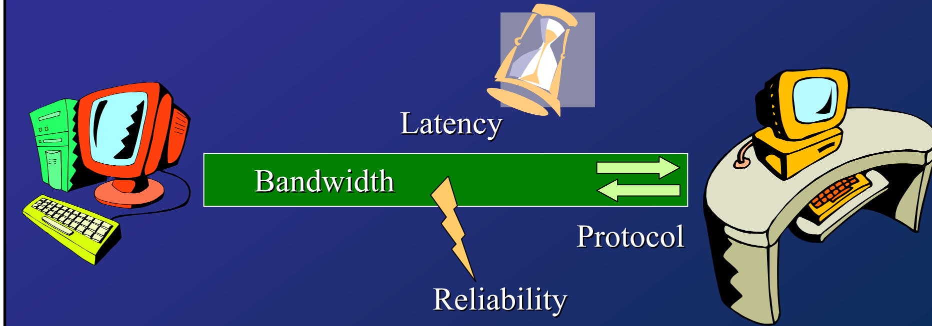


- Network bandwidth
  - the rate at which the network can deliver data to the destination host (bits per second, bps)
- Network reliability
  - a measure of how much data is lost by the network during the journey from source to destination host
  - types of data loss:
    - dropping: the data does not arrive
    - corruption: the content has been changed

Fundamentals of Data Transfer 3 (3)

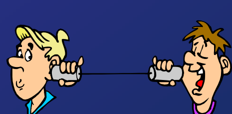
- Network protocol
  - a set of rules that two applications use to communicate with each other
  - packet formats
    - understanding what the other endpoint is saying
  - packet semantics
    - what the recipient can assume when it receives a packet
  - error behaviour
    - what to do if (when) something goes wrong

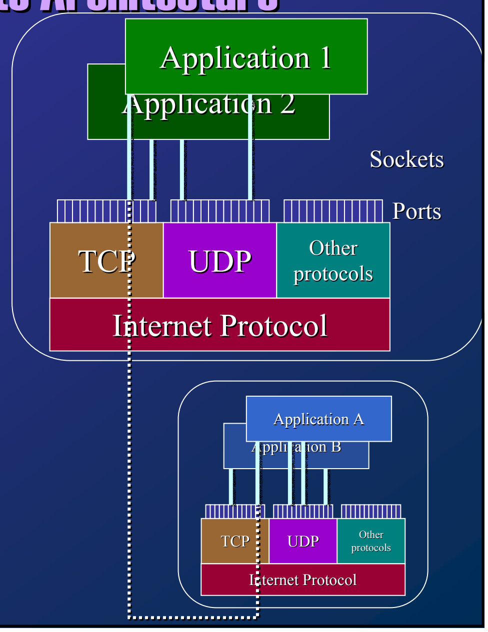
Network Communication



The BSD Sockets Architecture

- A socket is a software representation of the endpoint to a communication channel
- Reliable/unreliable communication, single/multiple destinations, etc.
- Includes several pieces of information, such as
  - protocol
  - destination host and port
  - source host and port

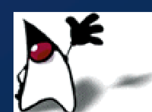




BSD = Berkeley Software Distribution

Sockets in Java

- Networking related classes are in the package `java.net`
- IP addresses are handled with the `InetAddress` class
  - Creation (note: no constructor):
    `InetAddress address = InetAddress.getByName(`*address*`);`
  - Parameter *address*
    - in DNS format (`"staff.cs.utu.fi"`)
    - as an IP number (`"139.232.75.8"`)
    - `null` (= `"localhost"` = `"127.0.0.1"`)
- Port numbers 1–1024 are reserved
- Socket types:
  - `ServerSocket`: handles connection requests directed to a given port
  - `Socket`: actual socket which handles the communication



Socket Example: The Code

Server

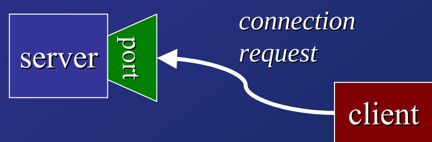
```
ServerSocket s =
    new ServerSocket(PORT);
try {
    Socket socket = s.accept();
    try {
        // Use the socket.
    } catch (IOException e) {
        // Transfer failed.
    } finally {
        socket.close();
    }
} catch (IOException e) {
    // Connection failed.
} finally {
    s.close();
}
```

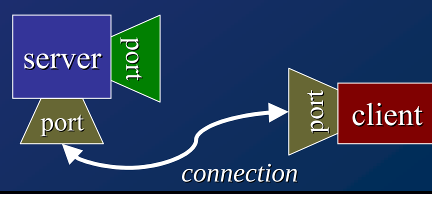
Client

```
try {
    Socket socket = new
        Socket(address, PORT);
    try {
        // Use the socket.
    } catch (IOException e) {
        // Transfer failed.
    } finally {
        socket.close();
    }
} catch (IOException e) {
    // Connection failed.
}
```

Socket Example: What Happens

- Server creates `ServerSocket` which begins to listen to the given port
- The execution halts in the `accept` method call, until there is a connection request
- Client creates `Socket` with the server's address and the port number of the server socket
- Client's socket sends a connection request
- Server socket answers the reqest by creating `Socket` to *any* available port
- Server socket sends the port number of the new socket to the client
- Client's socket connects to the new socket
- In the client, the socket's constructor finishes
- In the server, the `accept` method returns the new socket.





Using Streams with Sockets

- Input stream:

```
BufferedReader in =
  new BufferedReader(
    new InputStreamReader(
      socket.getInputStream()));
```

- Output stream:

```
PrintWriter out =
  new PrintWriter(new BufferedWriter(
    new OutputStreamWriter(
      socket.getOutputStream())), true);
```

- Reading and writing as normal:
  - `out.println("foo");`
  - `String s = in.readLine();`
- Streams use TCP, which is reliable but slow