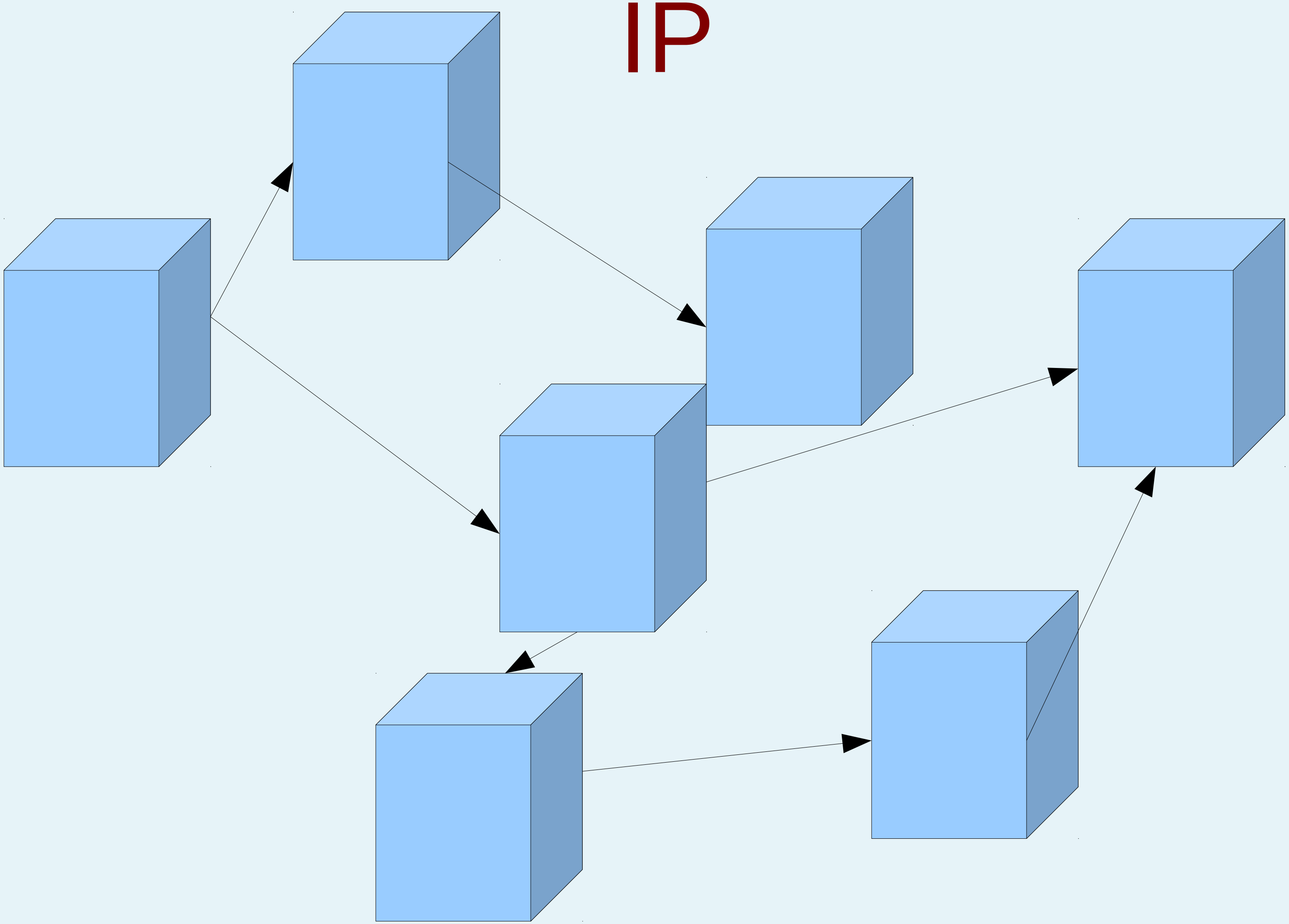
Game Networking

CS381 Spring 2012

Internet

- An information superhighway
- A collection of pipes
- Arpanet
    - Robust communication in the face of infrastructure breakdown
        - Packets instead of stream
        - Routers send packets towards destination
        - Incomplete knowledge of route to destination
    - Internet protocol - IP

IP



TCP and UDP over IP

- Connection based
- Guaranteed and Reliable
- Automatically packetizes
- Flow control
- Easy to use

- No concept of connection
- No guarantee of reliability or packet ordering
- Programmer packetizing
- Programmer flow control
- Programmer needs to handle lost packets

Sockets

- Network programming based on sockets
  - Open socket
  - Send on socket
  - Receive from socket

Socket code

- Send Thread
- While (true):
  - Get data
  - Send data on socket
- No blocking

- Receive Thread
- While (true):
  - Receive data from socket
  - Process data
- Blocked on receive

Python code

```
# TCP server example
import socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("", 5000))
server_socket.listen(5)
print "TCPServer Waiting for client on port 5000"
while 1:
        client_socket, address = server_socket.accept()
        print "I got a connection from ", address
        while 1:
                data = raw_input ( "SEND( TYPE q or Q to Quit):" )
                if (data == 'Q' or data == 'q'):
                        client_socket.send (data)
                        client_socket.close()
                        break;
                else:
                        client_socket.send(data)

                data = client_socket.recv(512)
                if ( data == 'q' or data == 'Q'):
                        client_socket.close()
                        break;
                else:
                        print "RECIEVED:" , data
```

Python code (Receive)

```
# TCP client example
import socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(("localhost", 5000))
while 1:
    data = client_socket.recv(512)
    if ( data == 'q' or data == 'Q'):
        client_socket.close()
        break;
    else:
        print "RECIEVED:" , data
        data = raw_input ( "SEND( TYPE q or Q to Quit):" )
        if (data <> 'Q' and data <> 'q'):
            client_socket.send(data)
        else:
            client_socket.send(data)
            client_socket.close()
            break;
```

UDP Server

```
# UDP server example
import socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(("", 5000))

print"UDPServer Waiting for client on port 5000"

while 1:
        data, address = server_socket.recvfrom(256)
        print "( " ,address[0], " " , address[1] , " ) said : ", data
```

UDP Client

```
# UDP client example
import socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while 1:
        data = raw_input("Type something(q or Q to exit): ")
        if (data <> 'q' and data <> 'Q'):
                client_socket.sendto(data, ("localhost",5000))
        else:
                break
client_socket.close()
```

Action Game Networking

- TCP not suitable
  - We are interested in most recent game state more than in reliably receiving game state
  - If there is network congestion TCP/IP may make congestion worse and worse with lots of resending of lost packets and acknowledgements

UDP Game networking

- Fixed sized packets
- Screen to find players and make game
  - Or broadcast on local net for local net game
- Specify authoritative server
- Ensure no possibility of cheating
  - Encryption
- Design Game networking protocol

381 engine

- Packet size: 65536
- Server broadcasts state every 100 ms
- No encryption
- Protocol
    - Server
    - Client

Breakout

Protocol for openEcslent

- Server
  - Broadcast state every 100 ms
  - Service client requests

- Client
  - Receive msg
  - Update state
  - Send user interaction

Server

- Sender Thread
  - Broadcast state
  - Broadcast send queue
  - Sleep 100 ms

- Receiver Thread
  - Receive msg
  - Store in receive queue

- Network Manager – every tick
  - Process receive queue
  - Put requested data in send queue

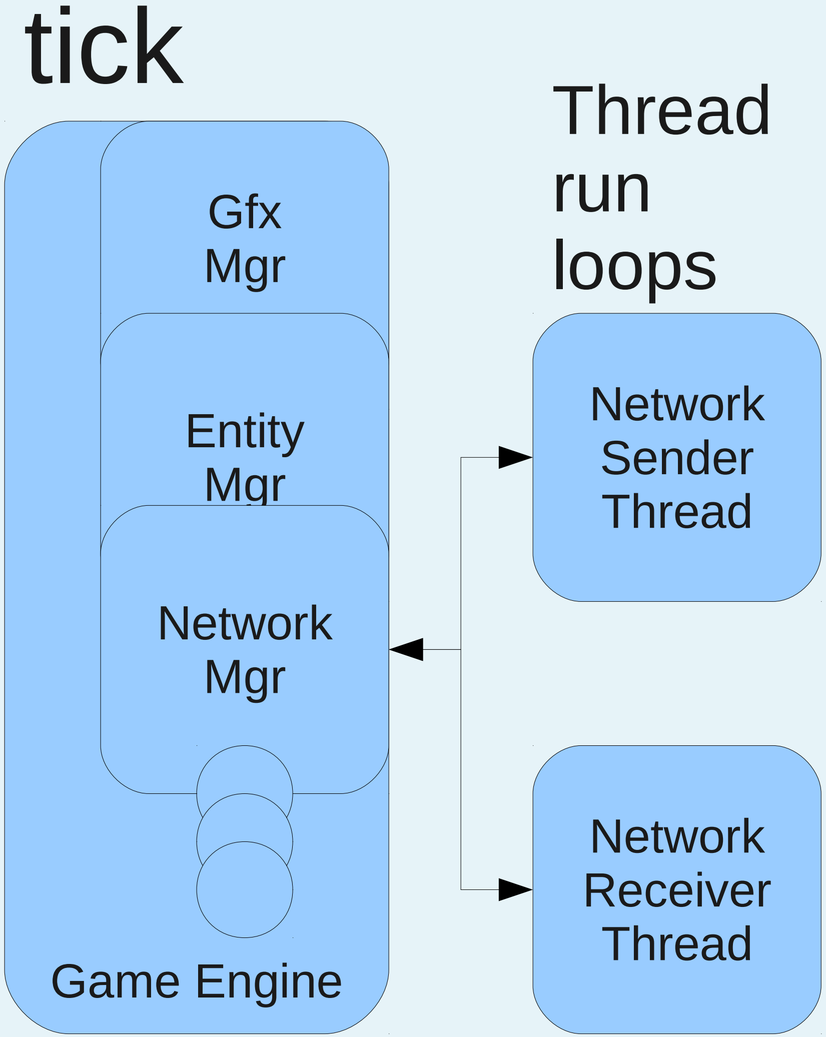
Client

- Sender Thread
  - Send all messages in Send Q
  - Sleep 100 ms

- Receiver Thread
  - Receive message
  - Put message in Receive Q

- Network Manager – every tick
  - Process all messages in Receive Q
  - Put new messages in Send Q

Big picture


tick

Thread
run
loops

Details

Read code!