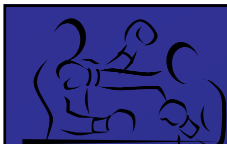
Internet Protocol (IP)

- Low-level protocols used by hosts and routers
- Guides the packets from source to destination host
- Hides the transmission path
  - phone lines, LANs, WANs, wireless radios, satellite links, carrier pigeons,…
- Applications rarely use the IP directly but the protocols that are written on top of IP
  - Transmission Control Protocol (TCP/IP)
  - User Datagram Protocol (UDP/IP)



TCP versus UDP

Transmission Control Protocol (TCP/IP)

- Point-to-point connection
- Reliable transmission using acknowledgement and retransmission
- Stream-based data semantics
- Big overhead
  - data checksums
- Hard to 'skip ahead'

User Datagram Protocol (UDP/IP)

- Lightweight data transmission
- Differs from TCP
  - connectionless transmission
  - 'best-efforts' delivery
  - packet-based data semantics
- Packets are easy to process
- Transmission and receiving immediate
- No connection information for each host in the operating system
- Packet loss can be handled

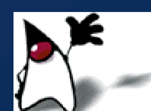
UDP and Datagrams in Java

- `DatagramSocket` can both send and receive packets
  - no server sockets because there is no need to establish a connection
- `DatagramPacket` includes all the data to be sent/received
  - maximum size 64 kB
- Constructing a receiving packet:

```
byte[] buffer = new byte[CAPACITY];
DatagramPacket dp1 =
            new DatagramPacket(buffer, CAPACITY);
```

- Constructing a packet to send:

```
byte[] message; // The bytes to send.
DatagramPacket dp2 =
            new DatagramPacket(message, message.length,
                               address, port);
```



Datagram Example

```
try {
    socket = new DatagramSocket(PORT);
    socket.receive(dp1);
    socket.send(dp2);
} catch (SocketException e) {
    // Could not open the socket.
} catch (IOException e) {
    // Problems with communication.
} finally {
    socket.close();
}
```

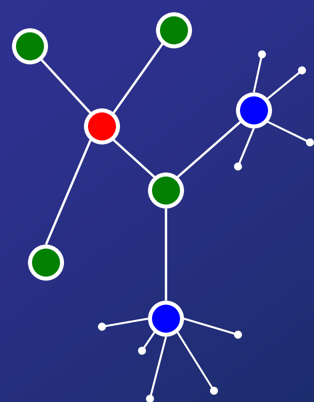
Datagram Contents

- Sender's address:
```
InetAddress addr = dp.getAddress();
```
- Sender's port:
```
int port = dp.getPort();
```
- Packet payload size:
```
int size = dp.getLength();
```
- Packet payload data:
```
byte[] data = dp.getData();
```

IP Broadcasting

- Using a single UDP/IP socket, the same packet can be sent to multiple destinations by repeating the send call
  - 'unicasting'
  - great bandwidth is required
  - each host has to maintain a list of other hosts
- IP broadcasting allows a single transmission to be delivered to all hosts on the network
  - a special bit mask of receiving hosts is used as a address
- With UDP/IP, the data is only delivered to the applications that are receiving on a designated port
- Broadcast is expensive
  - each host has to receive and process every broadcast packet
- Only recommended (and only guaranteed) on the local LAN
- Not suitable for Internet-based applications

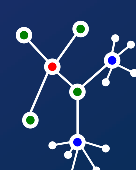
IP Multicasting 1 (3)



- Packets are only delivered to subscribers
- Subscribers must explicitly request packets from the local distributors
- No duplicate packets are sent down the same distribution path
- Original 'publisher' does not need to know all subscribers
- Receiver-controlled distribution

IP Multicasting 2 (3)

- 'Distributors' are multicast-capable routers
- They construct a multicast distribution tree
- Each multicast distribution tree is represented by a pseudo-IP address (multicast IP address, class D address)
  - 224.0.0.0–239.255.255.255
  - some addresses are reserved
  - local applications should use 239.0.0.0–239.255.255.255
- Address collisions possible
  - Internet Assigned Number Authority (IANA)
- Application can specify the IP time-to-live (TTL) value
  - how far multicast packets should travel
  - 0: to the local host
  - 1: on the local LAN
  - 2–31: to the local site (network)
  - 32–63: to the local region
  - 64–127: to the local continent
  - 128–254: deliver globally



IP Multicasting 3 (3)

- Provides desirable network efficiency
- Allows partitioning of different types of data by using multiple multicast addresses
- NVE participants can announce their presence by using application's well-known multicast address

- Older routers do not support multicasting
- Multicast-aware routers communicate directly by 'tunneling' data past the non-multicast routers (Multicast Backbone, Mbone)
  - Participant's local router has to be multicast-capable

Multicasting in Java

- Uses `DatagramPacket` as in UDP
- Sender sends datagram packets to a multicast address
- Receiver joins the multicast address (group):

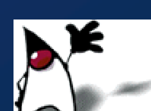
```
MulticastSocket socket =
    new MulticastSocket(PORT);
InetAddress group =
    InetAddress.getByName(GROUP_ADDRESS);
socket.joinGroup(group);
```

- Packets are received like normal UDP datagrams:

```
socket.receive(dp);
```

- Finally the receiver leaves the group and closes the socket:

```
socket.leaveGroup(group);
socket.close();
```



Multicast Example: Sender

```
class MulticastSender {
    private DatagramSocket socket;

    public MulticastSender() {
        try {
            socket = new DatagramSocket(PORT);
        } catch (SocketException e) { /* Construction failed. */
    }   }

    public void send(byte[] data) {
        try {
            Datagram packet = new DatagramPacket(data,
                 data.length, GROUP_ADDRESS, PORT);
            socket.send(packet);
        } catch (IOException e) { /* Sending failed. */
    }   }

    public void finalize() {
        if (socket != null) socket.close();
        super.finalize();
}   }
```

Multicast Example: Receiver

```
class MulticastReceiver {
    private MulticastSocket socket;
    public MulticastReceiver() {
        try {
            socket = new MulticastSocket(PORT);
            socket.joinGroup(GROUP_ADDRESS);
        } catch (IOException e) { /* Joining failed. */ }
    }
    public byte[] receive() {
        byte[] buffer = new byte[BUFFER_SIZE];
        DatagramPacket packet =
            new DatagramPacket(buffer, buffer.length);
        try {
            socket.receive(packet);
            return packet.getData();
        } catch (IOException e) { /* Receiving failed. */ }
        return null;
    }
    public void finalize() {
        if (socket != null) { socket.leaveGroup(); socket.close(); }
        super.finalize();
}   }
```

Selecting an NVE Protocol 1 (4)

- Multiple protocols can be used in a single system
- Not which protocol should I use in my NVE but which protocol should I use to transmit *this piece of information*?

- Using TCP/IP
  - reliable data transmission between two hosts
  - packets are delivered in order, error handling
  - relatively easy to use

  - point-to-point limits its use in large-scale NVEs
  - bandwidth overhead

Selecting an NVE Protocol 2 (4)

- Using UDP/IP
  - lightweight
  - offers no reliability nor guarantees the order of packets
  - packets can be sent to multiple hosts
  - deliver time-sensitive information among a large number of hosts

  - more complex services have to be implemented in the application
    - serial numbers, timestamps
  - recovery of lost packets
    - positive acknowledgement scheme
    - negative acknowledgement scheme
      - more effective when the destination knows the sources and their frequency
  - transmit a quench packet if packets are received too often

Selecting an NVE Protocol 3 (4)

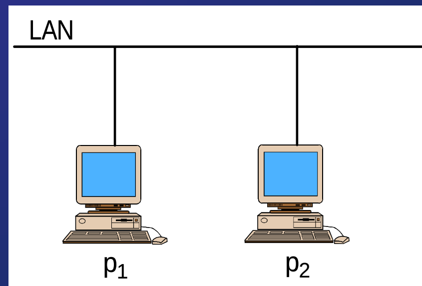
- Using IP broadcasting
  - design considerations similar to (unicast) UDP/IP
  - limited to LAN
  - not for NVEs with a large number of participants
  - to distinguish different applications using the same port number (or multicast address):
    - Avoid the problem entirely: assign the necessary number
    - Detect conflict and renegotiate: notify the participants and direct them to migrate a new port number
    - Use protocol and instance magic numbers: each packet includes a magic number at a well-known position
    - Use encryption

Selecting an NVE Protocol 4 (4)

- Using IP multicasting
  - provides a quite efficient way to transmit information among a large number of hosts
  - information delivery is restricted
    - time-to-live
    - group subscriptions
  - preferred method for large-scale NVEs
  - how to separate the information flows among different multicast groups
    - a single group/address for all information
    - several multicast groups to segment the information

Communication Architectures

- Logical connections
  - how the messages flow
- Physical connections
  - the wires between the computers
  - the limiting factor in communication architecture design



Two players on a LAN

Example: How May Players Can We Put into a Two-Player LAN?

- Distributed Interactive Simulation (DIS) protocol data unit (PDU): 144 bytes (1,152 bits)
- Graphics: 30 frames/second
- PDU rates
  - aircraft 12 PDU/second
  - ground vehicle 5 PDU/second
  - weapon firing 3 PDU/second
  - fully articulated human 30 PDU/second
- Bandwidth
  - Ethernet LAN 10 Mbps
  - modems 56 Kbps
- Assumptions:
  - sufficient processor power
  - no other network usage
  - a mix of player types

⇒ LAN: 8,680 packets/second
fully articulated humans + firing = 263 humans
aircrafts + firing = 578 aircrafts
ground vehicles + firing = 1,085 vehicles

- Typical NPSNET-IV DIS battle
  - limits to 300 players on a LAN
  - processor and network limitations

Example (cont'd)

⇒ Modem: 48 packets/second
fully articulated humans + firing = 1 human
aircrafts + firing = 3 aircrafts
ground vehicles + firing = 6 vehicles

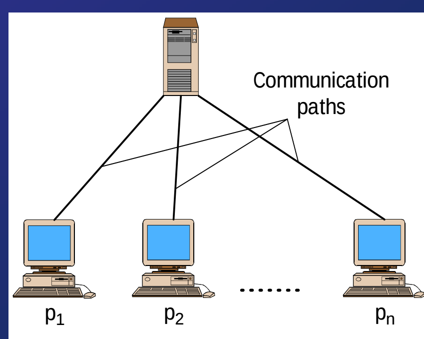
- Redesign packets
  - size 22%, 32 bytes

⇒ Modem: 218 packets/second
fully articulated humans + firing = 7 human
aircrafts + firing = 14 aircrafts
ground vehicles + firing = 27 vehicles

- In a two-player NVE on a LAN, the protocol selection (TCP, UDP, broadcast,...) hardly matters
- As the number of live or autonomous players increase an efficient architecture becomes more important

Multiplayer Client-Server Systems: Logical Architecture

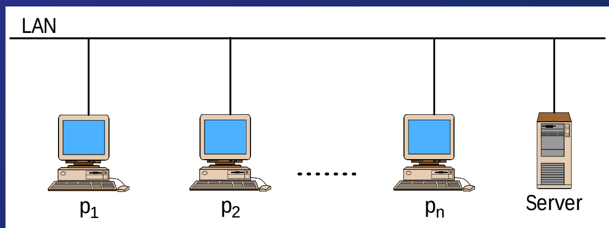
- Client-server system
  - each player sends packets to other players via a server
- Server slows down the message delivery
- Benefits of having a server
  - no need to send all packets to all players
  - compress multiple packets to a single packet
  - smooth out the packet flow
  - reliable communication without the overhead of a fully connected NVE
  - administration



Multiplayer client-server - logical architecture

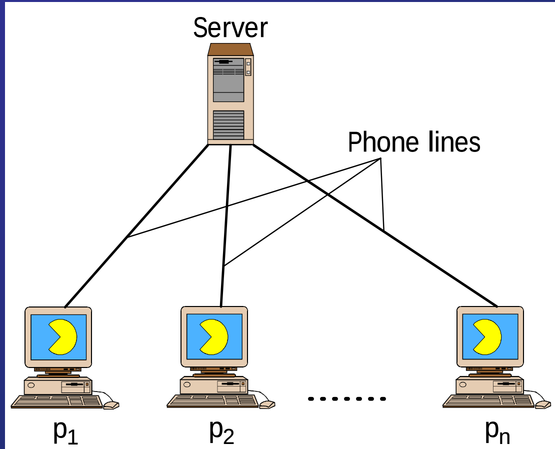
Multiplayer Client-Server Systems: Physical Architecture (on a LAN)

- All messages in the same wire
- Server has to provide some added-value function
  - collecting data
  - compressing and redistributing information
  - additional computation



Multiplayer client-server - physical architecture on a LAN

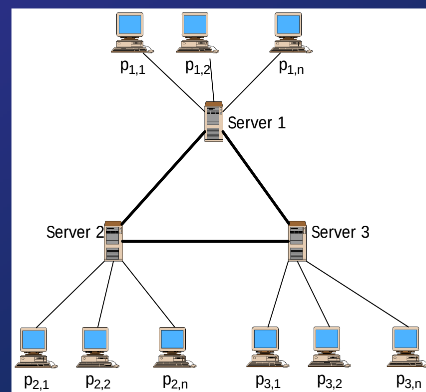
Physical Architecture Can Match the Logical Architecture

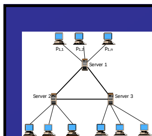


Multiplayer client-server - physical architecture with phone lines

Multiplayer Client-Server, with Multiple-Server Architectures

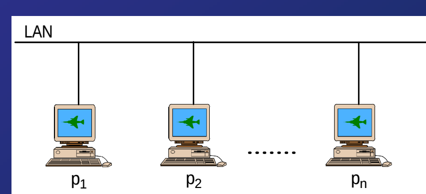
- Players can locate in the same place in the NVE, but reside on different servers
  - Real World ≠ Virtual World
- Server-to-server connections transmit the world state information
  - WAN, LAN
- Each server serves a number of client players
  - LAN, modem, cable modem
- Scalability



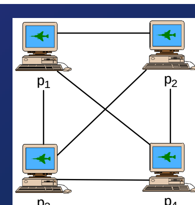


Peer-to-Peer Architectures

- In the *ideal* large-scale NVE design, avoid having servers at all
  - eventually we cannot scale out
  - a finite number of players
- Design goal
  - peer-to-peer communication
  - scalable within resources
- Peer-to-peer: communication goes directly from the sending player to the receiving player (or a set of them)

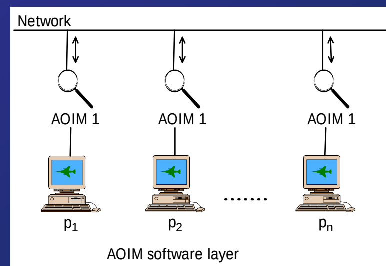


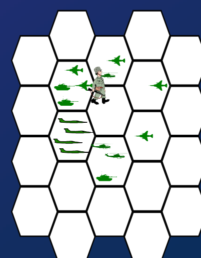
Peer-to-peer on a LAN



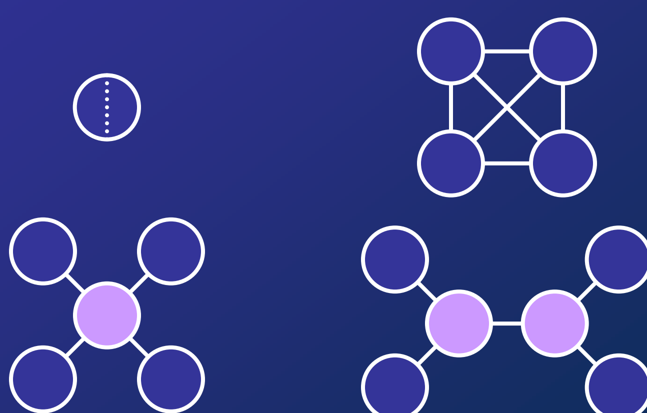
Peer-to-Peer with Multicast

- For a scalable NVE on a LAN, use multicast
- To utilize multicast, assign packets to proper multicast groups
- Area-of-interest management
  - assign outgoing packets to the right groups
  - receive incoming packets to the appropriate multicast groups
  - keep track of available groups
  - even out stream information





Basic Architecture Components



(Hint: This would be a good time to browse the additional literature.)