Lecture 11: Networking

CSE 40166 Computer Graphics

Peter Bui

University of Notre Dame, IN, USA

November 23, 2010

Networking in Computer Graphics?

Motivation

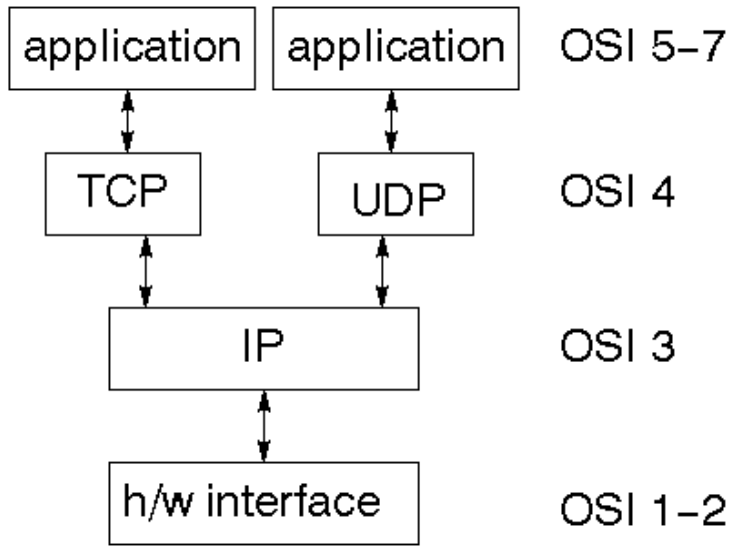
- Communication between multiple programs or components.
- Extend event programming loop.
- Integrate into application engine.

Examples

Networking Transport - Sockets

To communicate over across a network, we setup I/O devices called `sockets`:

- Bound to one or more `addresses` and typically one `port`.
- Communicate using `sockets` with different models:
  1. **Datagrams**: discrete data messages **(UDP)**.
  2. **Streams**: connection-based streaming I/O **(TCP)**.



Networking Transport - UDP

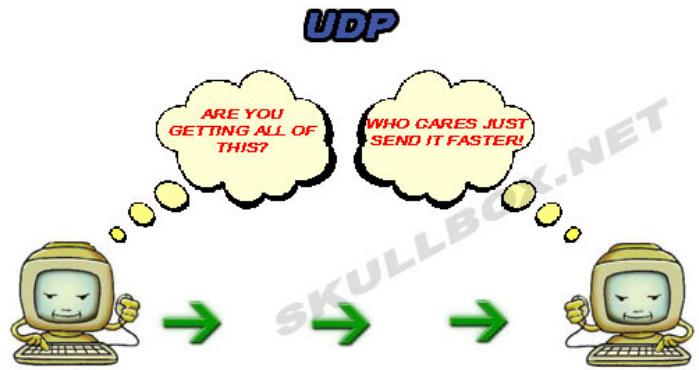
`UDP` is connectionless and unreliable:

Pros

- **Simple**: Just send and receive messages.
- **Minimal Overhead**: No handshakes, no buffering, no retries.
- **Multicast**: Send messages to a group of clients.

Cons

- **Unreliable**: Messages are not resent.
- **Unordered**: Messages are not automatically ordered.
- **Poor WAN Transport**: Limited to mostly LAN environments.



Networking Transport - TCP

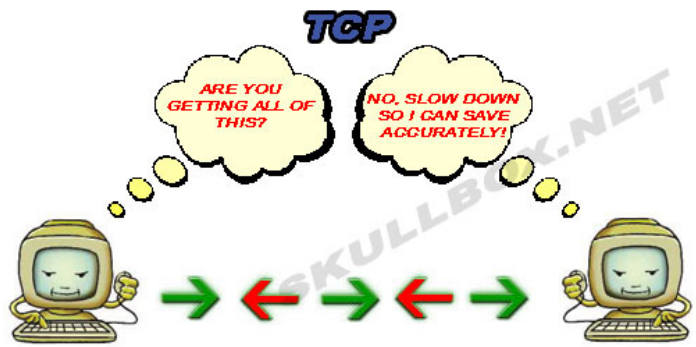
`TCP` provides a reliable connection-oriented protocol.

Pros

- **Streams**: sockets can be treated similar to `FILE *`.
- **Reliable**: scaling, buffering, ordering.

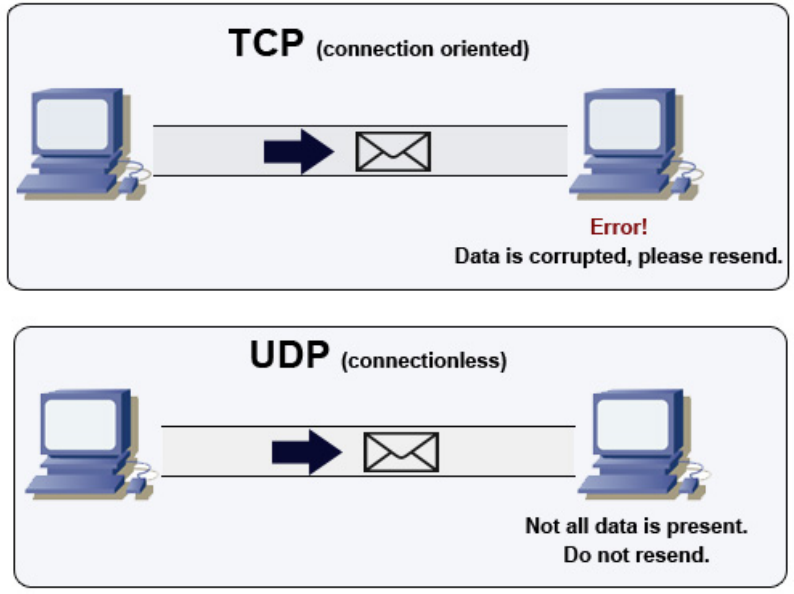
Cons

- **Latency**: *TANSTAAFL*.



Networking Transport - Summary

Most network games today use TCP or a custom protocol built on UDP that has many of TCP's features.



Network Architecture - Overview

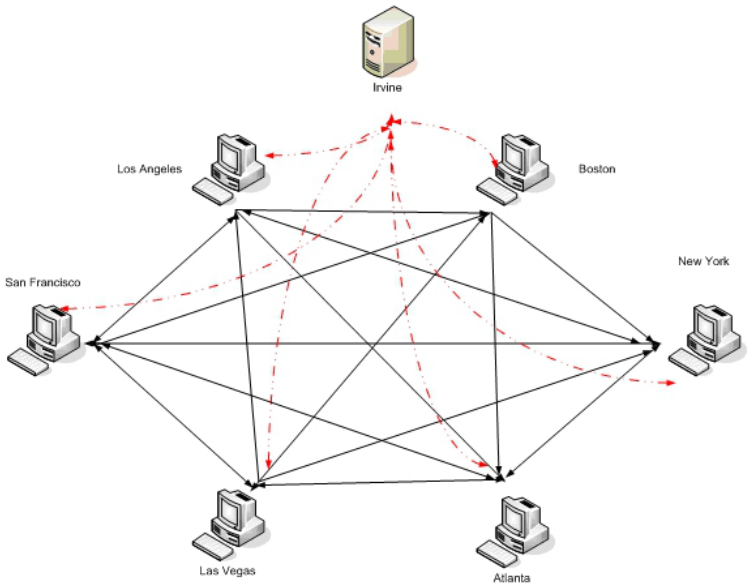
We need to organize our different software components in some sort of architecture, which answers the following questions:

1. **Who communicates with who?**
2. **Who has the game state?**
3. **Who breaks ties?**
4. **How do we deal with lag (latency)?**

There are two primary architectures:

1. **Peer-to-Peer**
2. **Client/Server**

Network Architecture - Peer-to-Peer



- Each machine is equal and maintains state of game.
- At each time step, each machine sends out its input and time.
- Each machine carries out the exact same logic on the exact same inputs.

Network Architecture - Peer-to-Peer (Continued)

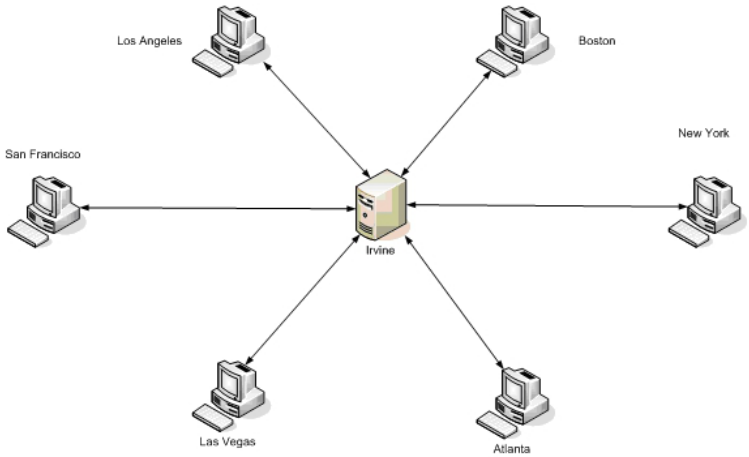
Advantages

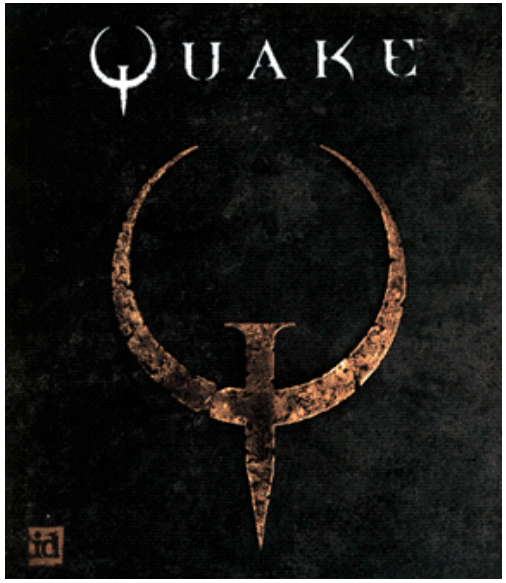
1. **No central bottleneck**: Workload is distributed more evenly across machines.
2. **Less laggy**: No blocking or waiting for network for local movement.

Problems

1. **Persistence**: All players have to start game together for consistent gaming.
2. **Scalability**: Game runs in lock-step, so intermittent networking failures can lead to problems.
3. **Locked in Frame Rate**: All players run at same internal frame rate, which is difficult with heterogenous machines.

Network Architecture - Client/Server





- Server is responsible for making all gameplay decisions.
- Clients send keystrokes, while server sends back lists of objects and simulation and predication logic.
- Client maintains subset of game state locally.

Network Architecture - Client/Server (Continued)

Advantages

1. **Consistency**: Harder to cheat, since there is a central authority.
2. **Throttling**: Can manage lag to make game smoother.

Problems

1. **Host Advantage**: Host has advantage and sets the rules.
2. **Complexity**: Need to implement additional components such as prediction logic, state management, dead reckoning.

Network Architecture - Summary

Most internet-scale games use `client/server` since consistency and anti-cheating is important. Local area network games may still use `peer-to-peer`, but this is rapidly going away.

Network Issues

Bandwidth

How **much** data can I send in `X` amount of time?

Latency

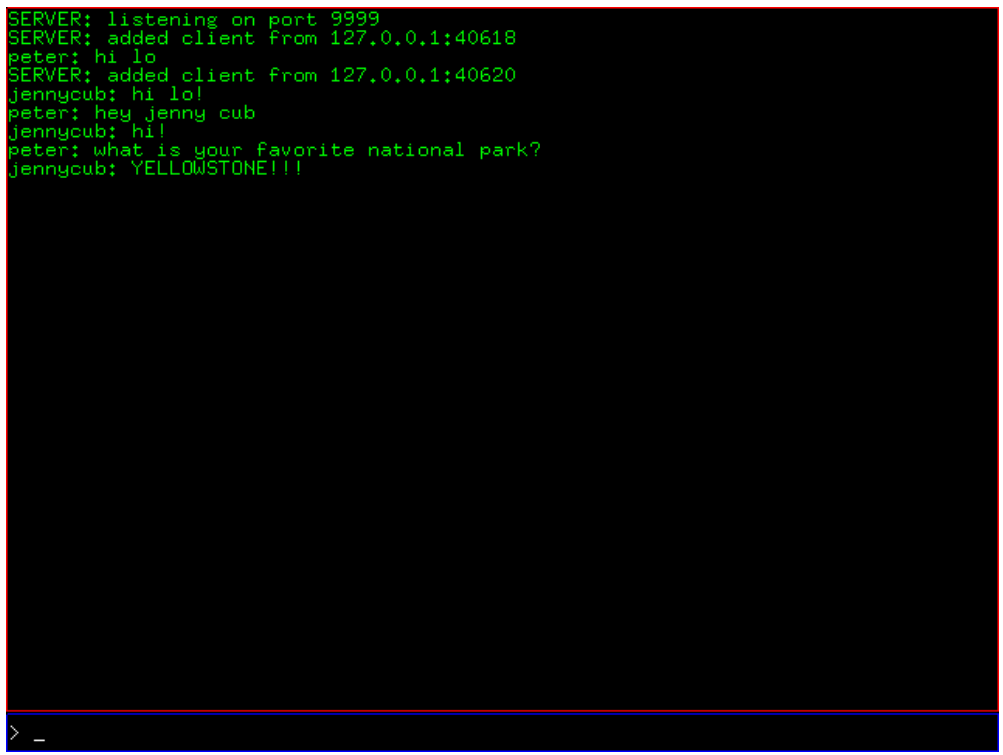
How **soon** will the data I send reach the destination?

For most games, **latency** is the most important and toughest issue to deal with. To help with **bandwidth** constraints, we can perform compression.

Example: OpenGL Chat

A simple OpenGL chat program:

1. Use client/server architecture (host is both client and server).
2. Use TCP communication (via CCTools Link library).
3. Use OpenGL event loop to handle communication and rendering.



Example: OpenGL Chat (Idle Function)

Use `glutIdleFunc` to check on network communcations:

```
1  int main(int argc, char *argv[])
2  {
3      // ..
4      glutIdleFunc(idle);
5      glutMainLoop();
6      // ..
7  }
```

```
1  void idle()
2  {
3      if (ChatServer) {    // Server role
4          if (!ServerLink)
5              server_listen();
6          else
7              server_poll_clients();
8      } else {             // Client role
9          if (!ClientLink)
10             client_connect();
11         else
12             client_read_messages();
13     }
14 }
```

Example: OpenGL Chat (Client Connect)

```
1  void client_connect()
2  {
3      char address[LINK_ADDRESS_MAX];
4      time_t stoptime;
5      int result;
6      char buffer[CHAT_LINE_MAX];
7
8      // Get address of hostname
9      result = domain_name_cache_lookup(ChatHostname, address);
10     if (!result) fatal("could not lookup name");
11
12     // Connect to host
13     stoptime   = time(NULL) + 10;
14     ClientLink = link_connect(address, ChatPort, stoptime);
15
16     // Check if we have connected
17     if (ClientLink) {
18         link_tune(ClientLink, LINK_TUNE_INTERACTIVE);
19         snprintf(buffer, CHAT_LINE_MAX,
20                  "connected to server at %s:%d", ChatHostname, ChatPort);
21         store_message("CLIENT", buffer, time(NULL));
22     }
23 }
```

Example: OpenGL Chat (Client Read Messages)

```
1  void client_read_messages()
2  {
3      int result;
4      char buffer[CHAT_LINE_MAX];
5      int argc;
6      char **argv;
7
8      // Check if we have any messages to read
9      if (link_usleep(ClientLink, 1000, 1, 0)) {
10         // Read messages
11         result = link_readline(ClientLink, buffer, CHAT_LINE_MAX, time(0) + 1.0);
12         if (result > 0) {
13             // Store messages in list
14             string_split_quotes(buffer, &argc, &argv);
15             if (argc == 3)
16                 store_message(argv[0], argv[1], atoi(argv[2]));
17             free(argv);
18         } else {
19             // Failed to read message, so disconnect
20             link_close(ClientLink);
21             ClientLink = NULL;
22             store_message("CLIENT", "disconnected from server", time(NULL));
23         }
24     }
25 }
```

Example: OpenGL Chat (Server Listen)

```
1  void server_listen()
2  {
3      char buffer[CHAT_LINE_MAX];
4
5      // Listen on port
6      ServerLink = link_serve(ChatPort);
7      if (!ServerLink) {
8          snprintf(buffer, CHAT_LINE_MAX,
9                   "could not listen on port %d: %s", ChatPort, strerror(errno));
10     } else {
11         snprintf(buffer, CHAT_LINE_MAX, "listening on port %d", ChatPort);
12     }
13
14     store_message("SERVER", buffer, time(NULL));
15
16     // Setup polling table
17     PollTable = malloc(sizeof(struct link_info *) * PollTableSize);
18 }
```

Example: OpenGL Chat (Server Poll Clients)

```
1  void server_poll_clients()
2  {
3      int n, result;
4
5      // Add master to polling table
6      PollTable[0].link    = ServerLink;
7      PollTable[0].events  = LINK_READ;
8      PollTable[0].revents = 0;
9
10     // Add clients to polling table
11     n = 1;
12     for (std::list< struct link *>::iterator i = Clients.begin();
13          i != Clients.end(); i++) {
14         PollTable[n].link    = (*i);
15         PollTable[n].events  = LINK_READ;
16         PollTable[n].revents = 0;
17         n++;
18     }
19
20     // Check connections for activity
21     result = link_poll(PollTable, n, 10);
22     if (result < 0) return;
23
24     // Add new clients
25     if (PollTable[0].revents) server_add_client();
26
27     // Read messages from clients
28     for (int i = 1; i < n; i++)
29         if (PollTable[i].revents) server_read_message(PollTable[i].link);
30
31     // Deliver outgoing messages
32     server_send_outgoing();
33 }
```

Example: OpenGL Chat (Server Add Client)

```
1  void server_add_client()
2  {
3      struct link *client_link;
4      char address[LINK_ADDRESS_MAX];
5      int port;
6      char buffer[CHAT_LINE_MAX];
7
8      // Try to accept new client
9      client_link = link_accept(ServerLink, time(NULL));
10     if (client_link) {
11         link_tune(client_link, LINK_TUNE_INTERACTIVE);
12
13         if (link_address_remote(client_link, address, &port)) {
14             snprintf(buffer, CHAT_LINE_MAX,
15                      "added client from %s:%d", address, port);
16             store_message("SERVER", buffer, time(NULL));
17         }
18
19         // Add to list of clients
20         Clients.push_back(client_link);
21     }
22 }
```

Example: OpenGL Chat (Server Read Message)

```
1  void server_read_message(struct link *client_link)
2  {
3      char buffer[CHAT_LINE_MAX];
4
5      // Try to read in new message
6      if (link_readline(client_link, buffer, CHAT_LINE_MAX, time(NULL) + 1)) {
7          int argc;
8          char **argv;
9
10         // Store message
11         string_split_quotes(buffer, &argc, &argv);
12         if (argc == 3) {
13             store_message(argv[0], argv[1], atoi(argv[2]));
14             store_outgoing(argv[0], argv[1], atoi(argv[2]));
15         }
16         free(argv);
17     } else {
18         // Read failed, so disconnect client
19         link_close(client_link);
20         Clients.erase(std::find(Clients.begin(), Clients.end(), client_link));
21     }
22 }
```

Example: OpenGL Chat (Server Send Outgoing)

```
1  void server_send_outgoing()
2  {
3      char buffer[CHAT_LINE_MAX];
4      int result;
5
6      // For each outgoing message
7      while (!Outgoing.empty()) {
8          struct message m = Outgoing.front();
9
10         // Send to each connected client
11         for (std::list< struct link *>::iterator i = Clients.begin();
12              i != Clients.end(); i++) {
13             snprintf(buffer, CHAT_LINE_MAX,
14                      "\"%s\" \"%s\" %ld\n", m.author, m.message, time(NULL));
15             result = link_write(*i, buffer, strlen(buffer), time(NULL) + 1);
16             if (result != (int)strlen(buffer)) {
17                 // Send failed, so disconnect
18                 link_close(*i);
19                 i = Clients.erase(i);
20             }
21         }
22
23         free((char *)m.author);
24         free((char *)m.message);
25         Outgoing.pop();
26     }
27 }
```

Example: OpenGL Chat (Summary)

- Hook into OpenGL event loop through idle or timer callback.
- In each callback, perform the following:
  - **Server**:
    1. Listen on specified port.
    2. Accept incoming client connections.
    3. Read incoming messages.
    4. Send outgoing messages.
  - **Client**:
    1. Connect to server.
    2. Read incoming messages.
- Link failures lead to disconnects.
- Poll on links to avoid blocking for too long.

Resources

- **Link Library API**
  http://cse.nd.edu/~ccl/software/manuals/api/html/link_8h.html
- **Unreal Networking Architecture**
  http://unreal.epicgames.com/Network.htm
- **Dead Reckoning: Latency Hiding for Networked Games**
  http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php
- **Targeting: A Variation of Dead Reckoning**
  http://www.gamedev.net/reference/articles/article1370.asp
- **Latency Compensating Methods**
  http://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization