Web Applications
Chris Piech
CS106A, Stanford University
For the first time ever in CS106A
[suspense]
Send reports via:
mobile phone :: email :: web
Final Assignment

Shahidi
A CS 106A Production

- Hovering over the Stanford Oval
  37.0, -122.0

- Going for a swim
  34.0, -126.0

- At the Hollywood Sign
  34.0, -118.0

Piech, CS106A, Stanford University
Usahidi

SteamTunnelServer

Starting server on port 8000...
addProfile (name=Mehran) => success
addProfile (name=Chris) => success
addProfile (name=Chris) => Error: Database already contains Chris.
getStatus (name=Chris) => none
setStatus (name=Chris, status=teaching) => success
getStatus (name=Chris) => teaching
addFriend (name2=Mehran, name1=Chris) => success
getFriends (name=Chris) => [Mehran]
addProfile (name=Julie) => success
getImg (name=Julie) => none
getStatus (name=Julie) => none
getFriends (name=Julie) => []
setImg (img=Julie2.jpg, name=Julie) => success
getImg (name=Julie) => Julie2.jpg
getStatus (name=Julie) => none
getFriends (name=Julie) => []
addFriend (name2=Chris, name1=Julie) => success
getImg (name=Julie) => Julie2.jpg
getStatus (name=Julie) => none

SteamTunnelClient

Chris

Friends
Mehran
Julie

Chris is teaching

Julie added as a friend.
Ushahidi
(testimony)
Ushahidi
(testimony)
Shahidi

(whitness)
The Name

The idea that governs Facebook was created, then shut down, at Stanford years before Mark Zuckerberg appeared.

Silicon Valley’s Sun Microsystems, Cisco, Hewlett-Packard, Yahoo, Google and Facebook are some of the largest technology companies in the world. Stanford alumni founded the first five. But if things had turned out a bit differently in the fall of 1999, would we have been able to attribute the creation of Facebook to Stanford as well?
Question

How could users join different SteamTunnels but still connect to one another.

Piech, CS106A, Stanford University
Review
Background: The Internet

The internet is just many programs sending messages (as Strings)

Thanks Nick for the teaching YEAH
Background: The Internet

The internet is just many programs sending messages (as *Strings*)

Facebook datacenter

Your computer (facebook.com)
The internet is just many programs sending messages (as Strings).
The internet is just many programs sending messages (as Strings)

Thanks Nick for the teaching YEAH
Background: The Internet

The internet is just many programs sending messages (as Strings)

Facebook datacenter

“Server”

“Client”

Your computer (facebook.com)

“request”

“teaching YEAH”

“response”

Get status for “Nick Troccoli”

Thanks Nick for the teaching YEAH
SteamTunnel

SteamTunnelServer

“Server”

Your task

RESPONSE

REQUEST

SteamTunnelClient

“Client”

Extra credit

Your task

Thanks Nick for the teaching YEAH
Another way to get Server/Client

First, imagine a world before Server/Clients…

* This blob represents one program on one machine
Another way to get Server/Client

First, imagine a world before Server/Clients…

* This blob represents one program on one machine
Another way to get Server/Client

First, imagine a world before Server/Clients…

* This blob represents one program on one machine
Another way to get Server/Client

Now our application runs across two programs

* Each blob represents one program on one machine.
Another way to get Server/Client

Which means many clients can connect to the data

* Each blob represents one program on one machine
The Internet

Client

Shahidi datacenter

“Server”

REQUEST

Tate Ole Keko’s computer (shahidi.com)

RESPONSE

REQUEST

Chris’ phone (shahidi app)

RESPONSE

REQUEST

Your mom’s computer (linux shell)

“Client”

“Client”

“Client”

“Client”
Most of the Internet

Server / Clients

Aka “the backend”

Aka “the cloud”

Aka “the frontend”

Aka “the GUI”
public String requestMade(Request request) {
    // server code goes here
}

// make a Server object
private SimpleServer server = new SimpleServer(this, 8000);

public void run() {
    // start the server
    server.start();
}
public String requestMade(Request request) {
    // server code goes here
}

// make a Server object
private SimpleServer server = new SimpleServer(this, 8000);

public void run(){
    // start the server
    server.start();
}
What is a Request?

/* Request has a command */
String command;

/* Request has parameters */
HashMap<String,String> params;

Request request

// methods that the server calls on requests
request.getCommand();
request.getParam(key);  //returns associated value
Requests are like Remote Method Calls

Server has a bunch of discrete things it can do

- addUser
- getStatus
Requests are like Remote Method Calls
Requests are like Remote Method Calls

```java
request.getCommand();
=> "addUser"
```
Your Server Code

```java
/**
 * Starts the server running so that when a program sends
 * a request to this computer, the method requestMade is
 * called.
 */

public void run() {
    println("Starting server on port " + PORT);
    server.start();
}

/**
 * When a request is sent to this computer, this method
 * called. It must return a String.
 */

public String requestMade(Request request) {
    String cmd = request.getCommand();
    println(request.toString());

    // your code here.

    return "Error: Unknown command " + cmd + ".";
}

Piech, CS106A, Stanford University
```
Where we left off...
There are two types of internet programs. Servers and Clients
Now, the client
A Clients’s Purpose

1. Interact with the user
2. Get data from its server
3. Save data to its server
try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

} catch(IOException e) {

    // The internet is a fast and wild world my friend

}
try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

} catch (IOException e) {

    // The internet is a fast and wild world my friend

}
try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

} catch(Exception e) {

    // The internet is a fast and wild world my friend

}
try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

} catch(IOException e) {

    // The internet is a fast and wild world my friend

}
try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

} catch(IOException e) {

    // The internet is a fast and wild world my friend

}
try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

} catch(IOException e) {

    // The internet is a fast and wild world my friend

}
try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

    } catch (IOException e) { 

        // The internet is a fast and wild world my friend

    }


try {

    // 1. construct a new request
    Request example = new Request("getStatus");

    // 2. add parameters to the request
    example.addParam("name", "chris");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, example);

} catch (IOException e) {

    // The internet is a fast and wild world my friend

}
Chat Server and Client

> CJP: Testing
> LMS: Hellooooo
> LMS: Im online!
> LMS: This is great
> LMS: And this is going to make it into lecture
```java
history = [
]

addMsg
msg = C: Hello world
```
history = [
  C: Hello world
]

getMsgs
index = 0
Chat Server

```
addMsg
msg = text

getMsgs
index = startIndex
```
There are two types of internet programs. Servers and Clients
public class ChatServer extends ConsoleProgram implements SimpleServerListener {

    private static final int PORT = 8080;

    private SimpleServer server = null;

    /* The server database is an ArrayList of Strings */
    private ArrayList<String> messages = new ArrayList<String>();

    public void run() {
        setFont("Courier-24");
        println("Starting server on port "+PORT+"...");
        server = new SimpleServer(this, PORT);
        server.start();
    }

... that’s not all
public class ChatServer extends ConsoleProgram implements SimpleServerListener {

    private static final int PORT = 8080;

    private SimpleServer server = null;

    /* The server database is an ArrayList of Strings */
    private ArrayList<String> messages = new ArrayList<String>();

    public void run() {
        setFont("Courier-24");
        println("Starting server on port "+PORT+"...");
        server = new SimpleServer(this, PORT);
        server.start();
    }

    ... that’s not all
Chat Server

```java
public class ChatServer extends ConsoleProgram
implements SimpleServerListener {

    private static final int PORT = 8080;

    private SimpleServer server = null;

    /* The server database is an ArrayList of Strings */
    private ArrayList<String> messages = new ArrayList<String>();

    public void run() {
        setFont("Courier-24");
        println("Starting server on port "+PORT+"...");
        server = new SimpleServer(this, PORT);
        server.start();
    }

    ... that’s not all
```
Chat Server

```java
public class ChatServer extends ConsoleProgram
implements SimpleServerListener {

    private static final int PORT = 8080;

    private SimpleServer server = null;

    /* The server database is an ArrayList of Strings */
    private ArrayList<String> messages = new ArrayList<String>();

    public void run() {
        setFont("Courier-24");
        println("Starting server on port "+PORT+"...\nserver = new SimpleServer(this, PORT);
server.start();
```

... that’s not all
public String requestMade(Request request) {
    println(request.toString());
    String command = request.getCommand();

    String result = "Error: Can't process request " + command;
    // we handle newMsg commands
    if(command.equals("newMsg")) {
        result = newMessage(request);
    }
    // we also handle getMsgs commands
    if(command.equals("getMsgs")) {
        result = getMessages(request);
    }

    println(" => " + result);
    return result;
}
public String requestMade(Request request) {
    println(request.toString());
    String command = request.getCommand();

    String result = "Error: Can't process request " + command;
    // we handle newMsg commands
    if(command.equals("newMsg")) {
        result = newMessage(request);
    }
    // we also handle getMsgs commands
    if(command.equals("getMsgs")) {
        result = getMessages(request);
    }

    println(" => " + result);
    return result;
}
public String requestMade(Request request) {
    println(request.toString());
    String command = request.getCommand();

    String result = "Error: Can't process request " + command;
    // we handle newMsg commands
    if(command.equals("newMsg")) {
        result = newMessage(request);
    }
    // we also handle getMsgs commands
    if(command.equals("getMsgs")) {
        result = getMessages(request);
    }

    println(" => " + result);
    return result;
}
public String requestMade(Request request) {
    println(request.toString());
    String command = request.getCommand();

    String result = "Error: Can't process request " + command;
    // we handle newMsg commands
    if(command.equals("newMsg")) {
        result = newMessage(request);
    }
    // we also handle getMsgs commands
    if(command.equals("getMsgs")) {
        result = getMessages(request);
    }

    println(" => " + result);
    return result;
}
HAPPY
CLIENT
FAQ
Question: This is cool Chris. But didn’t you just really dumb down servers?
Answer: No
Question: Why isn’t this in a browser?
“Native” vs “Browser”

Chat Client

Send hello from the browser

Messages

Refresh

> Chrome: hello from the browser
> Piech: hello from a native app

Chat Server

Starting server on port 8080...
getMsgs (index=0) => []
newMsg (msg=Chrome: hello from the browser) => success
getMsgs (index=0) => [Chrome: hello from the browser]
getMsgs (index=0) => [Chrome: hello from the browser]
newMsg (msg=Piech: hello from a native app) => success
getMsgs (index=1) => [Piech: hello from a native app]
getMsgs (index=1) => [Piech: hello from a native app]
Question: Localhost forever?
Use ngrok to get a url
Any security holes?
Want to learn more?

CS144 Computer Networking

Or CS193P

Or CS193A

Or CS108
Social Networks
Who do you love?

And how does Facebook know?
Your local social network

Piech, CS106A, Stanford University
You have some “groups”
But here is the love of your life

Your significant other

George

Piech, CS106A, Stanford University
Romantic Partnerships and the Dispersion of Social Ties: A Network Analysis of Relationship Status on Facebook

Lars Backstrom
Facebook Inc.

Jon Kleinberg
Cornell University

ABSTRACT
A crucial task in the analysis of on-line social-networking systems is to identify important people — those linked by strong social ties — within an individual’s network neighborhood. Here we investigate this question for a particular category of strong ties, those involving spouses or romantic partners. We organize our analysis around a basic question: given all the connections among a person’s friends, can you recognize his or her romantic partner from the network structure alone? Using data from a large sample of Facebook users, we find that this task can be accomplished with high accuracy, but doing so requires the development of a new measure of tie strength that we term ‘dispersion’ — the extent to which two people’s mutual friends are not themselves well-connected. The results offer methods for identifying types of structurally significant people in on-line applications, and suggest a potential expansion of existing theories of tie strength.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database applications—Data mining

Keywords: Social Networks; Romantic Relationships.

they see from friends [1], and organizing their neighborhood into conceptually coherent groups [23, 25].

Tie Strength.
Tie strength forms an important dimension along which to characterize a person’s links to their network neighbors. Tie strength informally refers to the ‘closeness’ of a friendship; it captures a spectrum that ranges from strong ties with close friends to weak ties with more distant acquaintances. An active line of research reaching back to foundational work in sociology has studied the relationship between the strengths of ties and their structural role in the underlying social network [15]. Strong ties are typically ‘embedded’ in the network, surrounded by a large number of mutual friends [6, 16], and often involving large amounts of shared time together [22] and extensive interaction [17]. Weak ties, in contrast, often involve few mutual friends and can serve as ‘bridges’ to diverse parts of the network, providing access to novel information [5, 15].

A fundamental question connected to our understanding of strong ties is to identify the most important person’s social network.


Piech, CS106A, Stanford University

October 2013
Dispersion: The extent to which two people’s mutual friends are not directly connected.
Dispersion: The extent to which two people’s mutual friends are not directly connected
filter bubble
fake news
hate speech
privacy
monopoly

Omer Reingold: Fairness Through
Computationally-Bounded Awareness
Question

How could users could join different SteamTunnels but still connect to one another.
Federated Internet Applications

Professor Monica Lam
The end.