Node Networking

2023 Fall

week 04 - class 09

Topics

- Node for Networking
  - Web Sockets
  - Socket.io

Node for Networking

net Module

- net module is used to create both servers and clients (called streams).
- It provides an asynchronous network wrapper.
  - net.Server class is used to create a TCP or local server
  - net.Socket is abstraction of TCP or local socket.

```
var net = require('net');
var server = net.createServer(function(c) { //'connection' listener
  console.log('server connected');
  c.on('end', function() {
    console.log('server disconnected');
  });
  c.write('hello\r\n');
  c.pipe(c);
});
server.listen(8124, function() { //'listening' listener
  console.log('server bound');
});
```

http module vs net module

- http.createServer() sets up a server that handles the HTTP protocol, which is indeed transmitted over tcp.

- net.createServer() creates a server that simply understands when a TCP connection has happened, and data has been transmitted
  - It doesn't know anything about whether a valid HTTP request has been received, etc.

- If you are writing a web server, favor http.createServer() over net.createServer()

HTTP

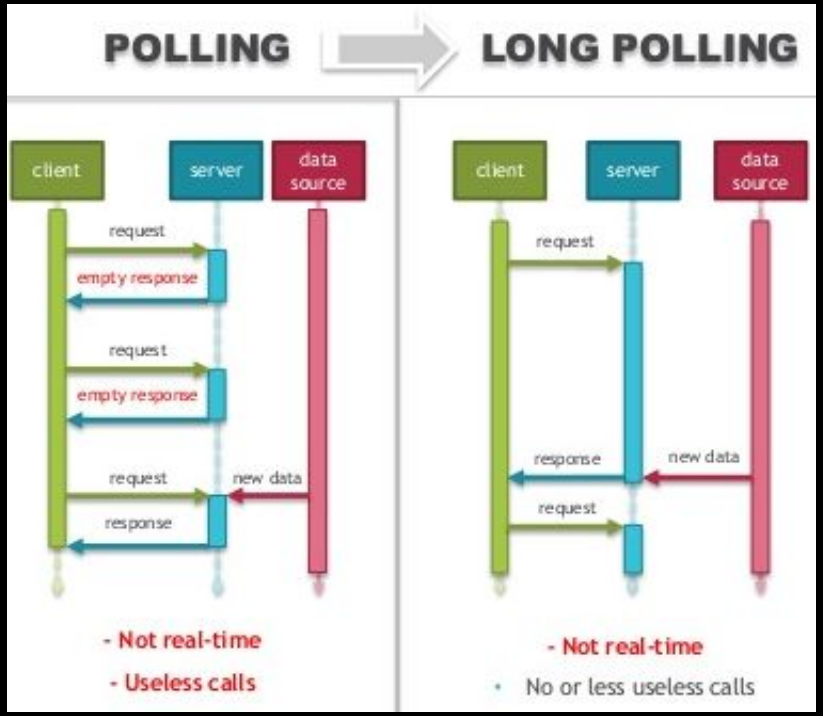
- HTTP (HyperText Transfer Protocol) is the basis for data communication on the internet
- The data communication starts with request sent from a client and ends with response received from a web server
- HTTP communication usually takes place over TCP/IP connections

Hidden costs of HTTP

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: openresty
Content-Length: 70488
Accept-Ranges: bytes
Date: Mon, 06 Feb 2017 03:10:26 GMT
Via: 1.1 varnish
Age: 63568
Connection: keep-alive
X-Served-By: cache-sjc3623-SJC
X-Cache: HIT
X-Cache-Hits: 1
X-Timer: S1486350626.819522,VS0,VE0
Vary: Accept-Encoding
```

- TCP handshake when establishing new connections
  - even worse for SSL
- HTTP headers on every message
  - always present, can vary in size and quantity
- For small messages, you may end up pushing around more HTTP headers than data!

HTTP Polling

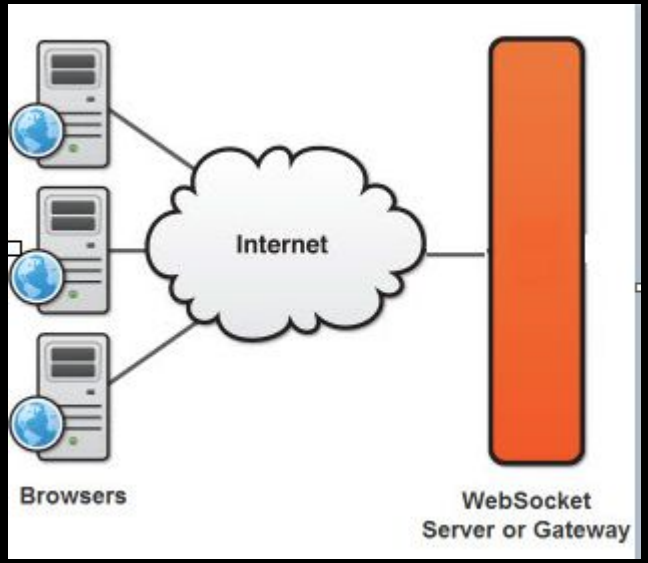


- There was no mechanism for the server to independently send or push data to client without the client first making a request
- HTTP polling to overcome this problem, where the client polls the server request new information

- Long Polling
  - Server waits until it has data to respond
  - Each request/response creates and closes a connection
  - Client has to wait to send new data until server responds

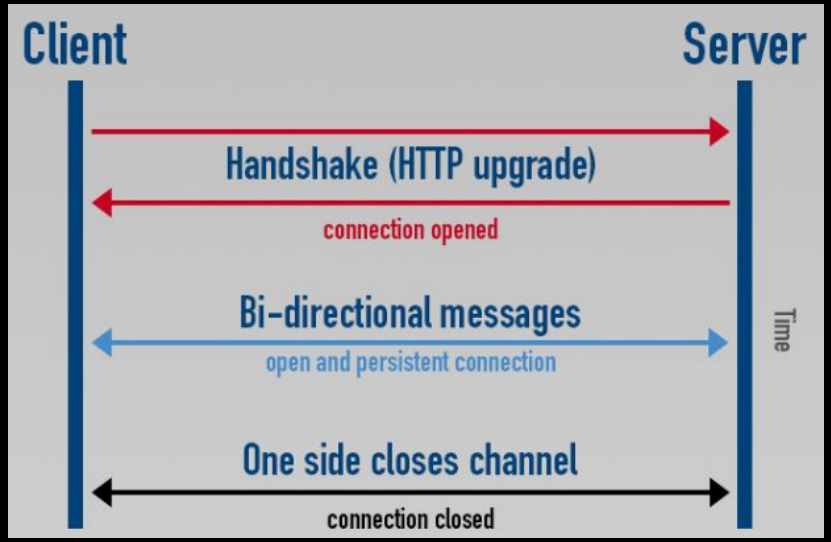
Web Sockets

WebSockets



- WebSockets protocol allows for two-way communication over a single tcp socket with a remote host
- Bi-directional
  - Client and server can send messages at any time
- Full duplex
  - Client and server can send updates at the same time
- Single long running connection with established context
- Effective use of bandwidth and CPU

WebSocket Connection



- Handshake
  - Client initiates connection
  - Server responds (accepts the upgrade)
- Once the WebSocket is established
  - both sides notified that socket is open
  - either side can send messages at any time
  - either side can close the socket

WebSocket Protocol Handshake

Client sends requests

request

```
GET /myapp HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: GhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Sec-WebSocket-Protocol: custom
Sec-WebSocket-Extensions: compress
Origin: http://example.com
...
```

Request upgrade to WebSocket connection

WebSocket handshake headers

Server sends response

response

```
HTTP/1.1 101 Switching Protocols
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: custom
Sec-WebSocket-Extensions: compress
```

Why use WebSockets?

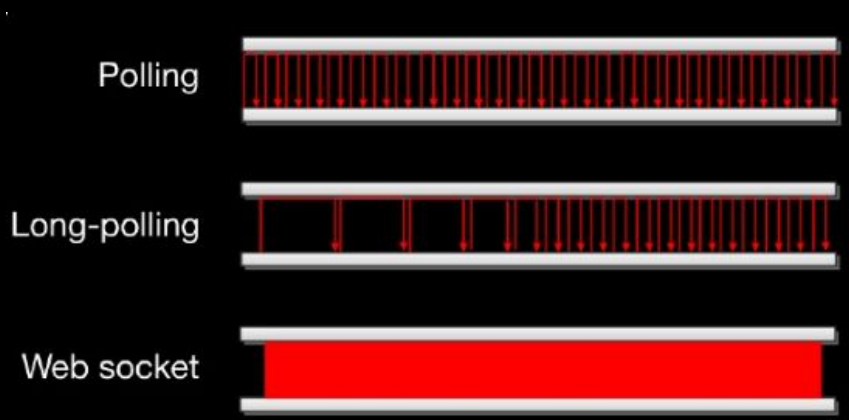
1. Real-time Applications

- Low latency 2-way communication for:
  - Gaming (Counter Strike, COD)
  - Collaboration (live wikis, google docs)
  - Dashboard (financial apps)
  - Tracking (watch user actions)
  - Presence (chat, instant messengers)

2. HTTP doesn't deliver

- HTTP hacks for real time
  - polling, long-polling, stream via hidden iframe
  - but these are slow, complex and bulky
- Rely on plugins:
  - Flash, Silverlight, Java applets
  - but these on

WebSockets vs HTTP Hacks



- Lower latency
  - no new TCP connections for each HTTP request
- Lower overhead for each message sent
  - 2 bytes vs. lines of HTTP header data
- Less traffic
  - since clients don't need to poll, messages only sent when we have data

Socket.IO

Socket.IO

https://socket.io

What?

- Real time application framework
- Wrapper around Websockets (browser + Node.js)
- Send events between the client and the server *(Publisher/Subscriber)*

Why?

- Fallback for old browsers (IE8+)
- JavaScript, native support for all devices ie. Android, iOS
- Trivial APIs

Socket.IO Main Features

- Reliability
  - connections are established even in presence of proxies, firewalls and antivirus software (via Engine.IO)
- Auto-reconnection support
  - unless instructed otherwise, a disconnected client will try to reconnect forever, until the server is available again
- Disconnection detection
  - a heartbeat mechanism is implemented at the Engine.IO level, allowing both the server and client to know when the other one is not responding anymore
- Binary support
  - any serializable data structure can be emitted

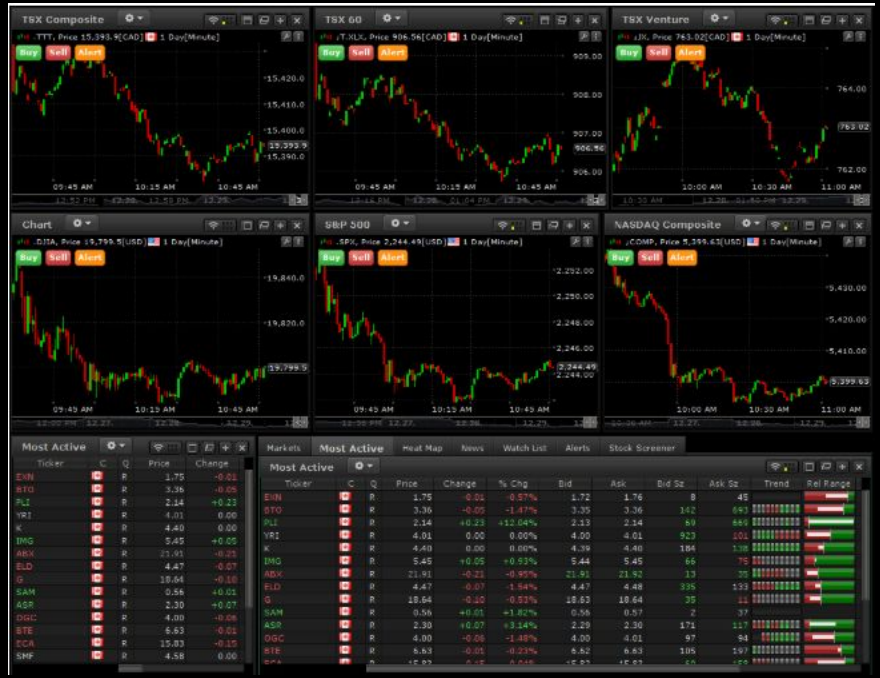
Socket.IO Main Features cont..

- Multiplexing support
  - create separate communication changes (Namespaces), but will share same underlying connection
- Room support
  - create arbitrary channels called Rooms, that sockets can join and leave. You can then broadcast to any given room, reaching every socket that joined it (ie. Private Chat)

Simple API connection

```
io.on('connection', function(socket){
  socket.emit('request', /* */); // emit an event to the socket
  io.emit('broadcast', /* */); // emit an event to all connected sockets
  socket.on('reply', function(){ /* */ }); // listen to the event
});
```

Socket.io Uses



- **Notifications**
  - Facebook and Twitter
- **Dashboards**
  - Real time Analytics
- **Group Connections**
  - Multiplayer Games
  - Trading, Sports Events + Gambling
  - Collaborative Forms ie. Google docs
- **Products**
  - Office, Yammer, Trello

Video - socket.io

https://youtu.be/UwS3wJoi7fY

Socket.IO Setup

```
npm install --save socket.io-client
```

Server (app.js)

```
var app = require('express')();
var server = require('http').Server(app);
var io = require('socket.io')(server);

server.listen(80);
// WARNING: app.listen(80) will NOT work here!

app.get('/', function (req, res) {
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
  socket.on('my other event', function (data) {
    console.log(data);
  });
});
```

Client (index.html)

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect('http://localhost');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
  });
</script>
```

What Socket.IO is not

- Socket.IO is NOT a WebSocket implementation
- It adds the following metadata to each packet
  - the packet type
  - the namespace
  - the ackId when a message acknowledgment is needed
- * A WebSocket client will not be able to successfully connect to a Socket.IO server and Socket.IO client will not be able to connect to a WebSocket server

Server API

Server:

- io.on('connection', callback(socket)) - new connected client

Socket:

- socket.on(event, callback(data)) - attach a new listener for the given event
- socket.emit(event, data) - send the event to this client
- socket.broadcast.emit(event, data) - send the event to all clients

Client API

Socket:

- socket.on(event, callback(data)) - attach a new listener for the given event
- socket.emit(event, data) - send the event to the server

....yes, that's it!

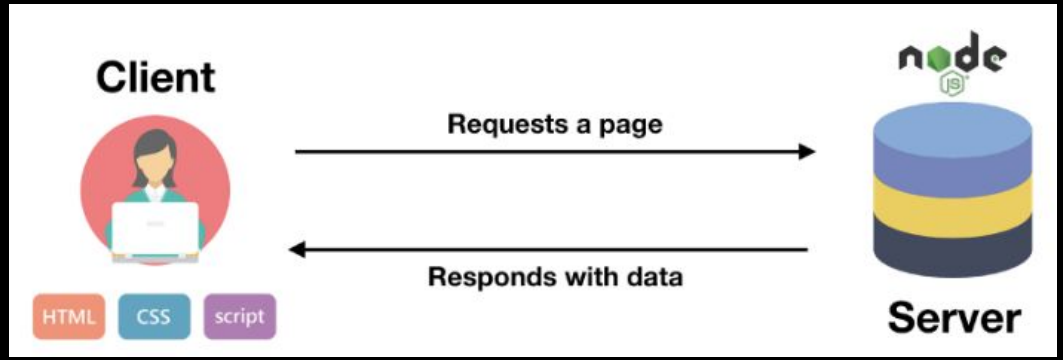
Building a Chat Application

Client Side

- EJS Template view engine
- HTML, CSS and JQuery
- Socket.io

Server Side

- Node.js *(Web Server)*
- Express API
- Socket.io



Chat Server

app.js

```
const express = require('express')
const app = express()


//set the template engine ejs
app.set('view engine', 'ejs')

//middlewares
app.use(express.static('public'))


//routes
app.get('/', (req, res) => {
    res.render('index')
})

//Listen on port 3000
server = app.listen(3000)
```

- The io object here will give us access to the socket.io library
- io object is now listening on each connection to our app.
- Each time a new user is connecting, it will print out "New user connected" in the console

```
//socket.io instantiation
const io = require("socket.io")(server)


//listen on every connection
io.on('connection', (socket) => {
    console.log('New user connected')

    // define socket event handlers here..
})
```

Chat Client

- Include the socket.io script reference in the EJS view file

```
<head>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.4/socket.io.js"></script>
  <title>Simple Chat App</title>
</head>
```

- When the client will load the file, it will automatically connect and create a new socket
- Add a key to the socket and "emit" to send data from client
- Add a socket listener on the server that will receive data with "on" event

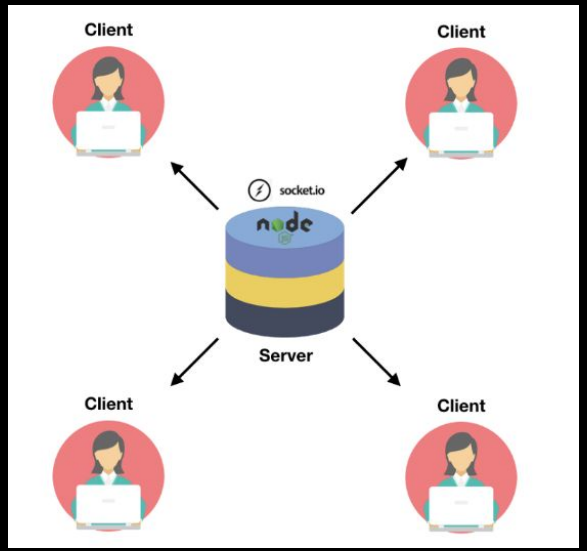
Client side

```
$(function(){
    //make connection
    var socket = io.connect('http://localhost:3000')


    socket.emit('hello', {message : "hello world! "})

}
```

Server side

```
//listen on every connection
io.on('connection', (socket) => {
    console.log('New user connected')


    //listen on hello
       socket.on('hello', (data) => {
        console.log(data);
    })
}
```

Multiple Sockets



- Each new Client represents a new socket connection
- We can define a socket to a namespace or a room

Send a message to all sockets connected using io.sockets.emit

```
io.sockets.emit('hi', 'everyone');
io.emit('hi', 'everyone'); // short form
```

You can call join to subscribe the socket to a given channel (room)

```
io.on('connection', function(socket){
  socket.join('some room');
});
```

Send a message to a given channel (room) using to or in

```
io.to('some room').emit('some event');
```