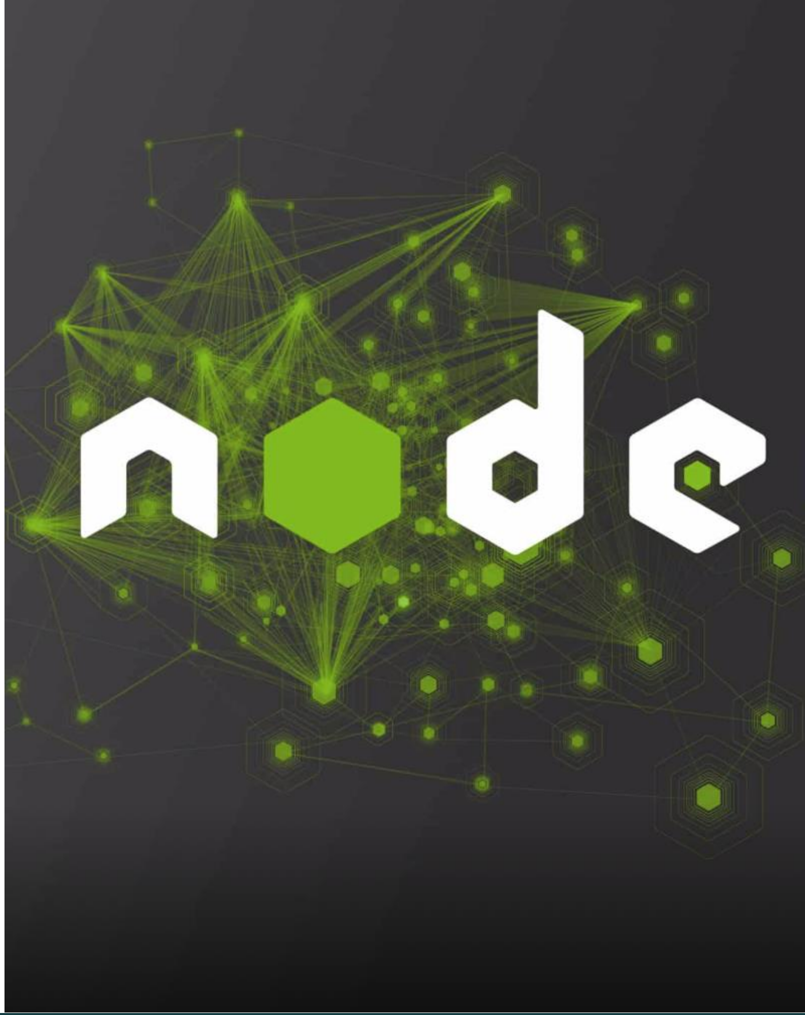
Node.js course’19

“net” module – asynchronous network API

CONFIDENTIAL | © 2019 EPAM Systems, Inc.

Agenda

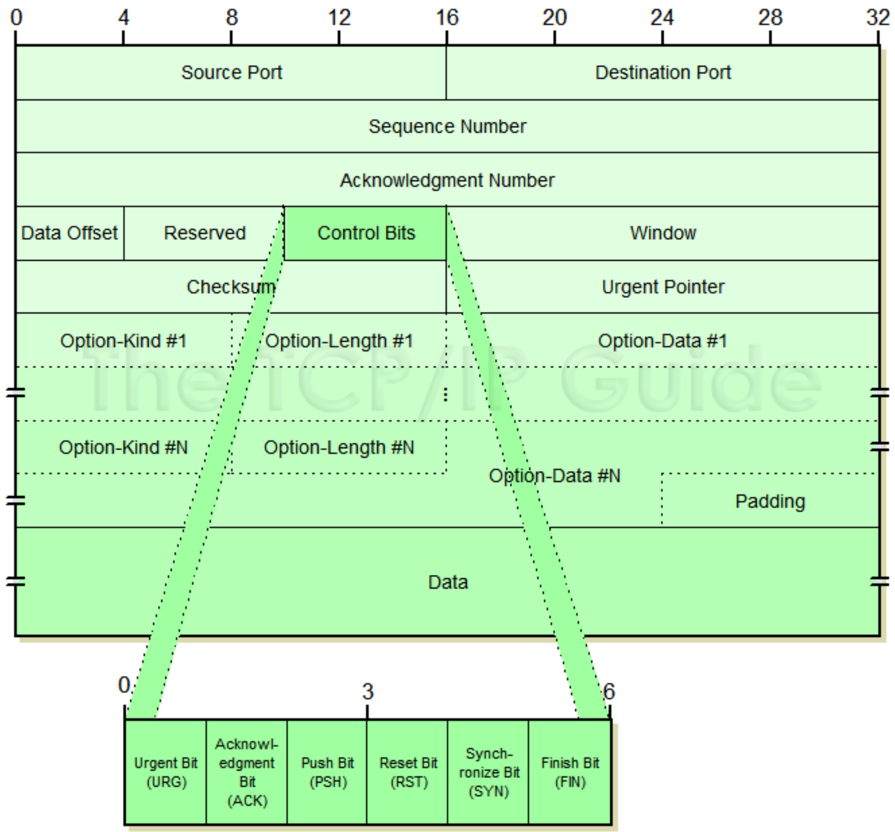


CONFIDENTIAL | © 2019 EPAM Systems, Inc.

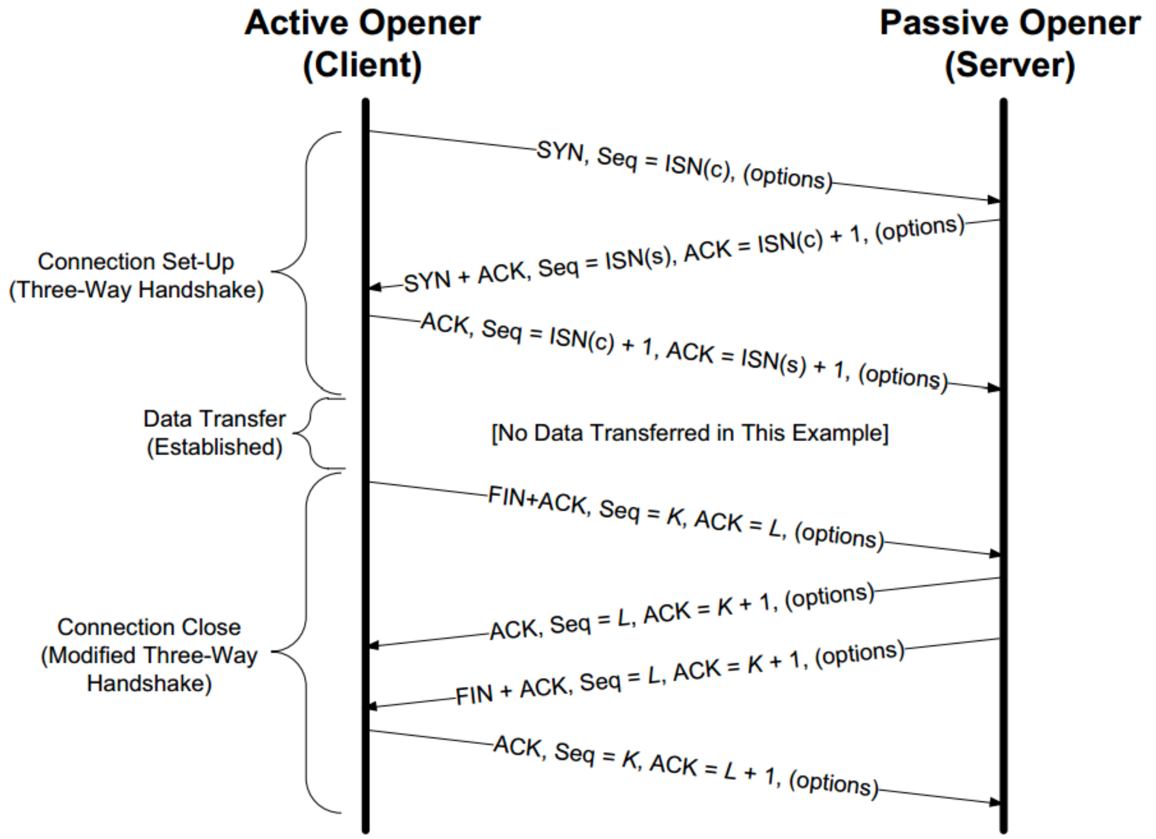
Open Systems Interconnection (OSI) model

Layer	Function	Example
Application (7)	Services used with the end user applications	HTTP/HTTPS, SMTP
Presentation (6)	Encrypt and decrypt (format) data	SSL, TLS
Session (5)	Establish&end connections between two hosts	NetBIOS, PPTP
Transport (4)	Transport protocol and error handling	TCP, UDP
Network (3)	Read the IP address from the data packet	Routers, Layer 3 switches
Data Link (2)	Read the MAC address from the data packet	Switches
Physical (1)	Send data on to the physical wire	Hubs, NICs, cables

Transmission Control Protocol (TCP)

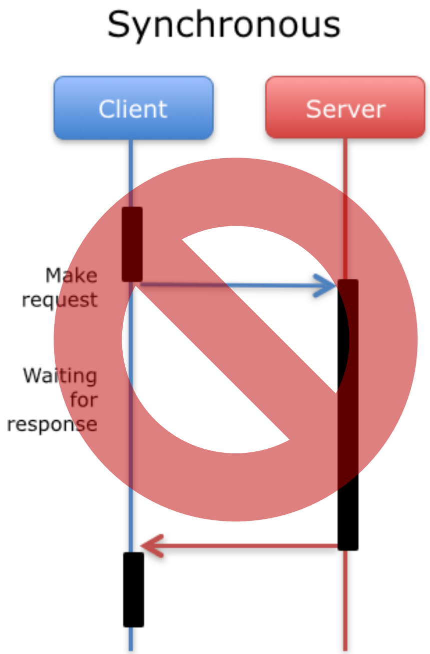


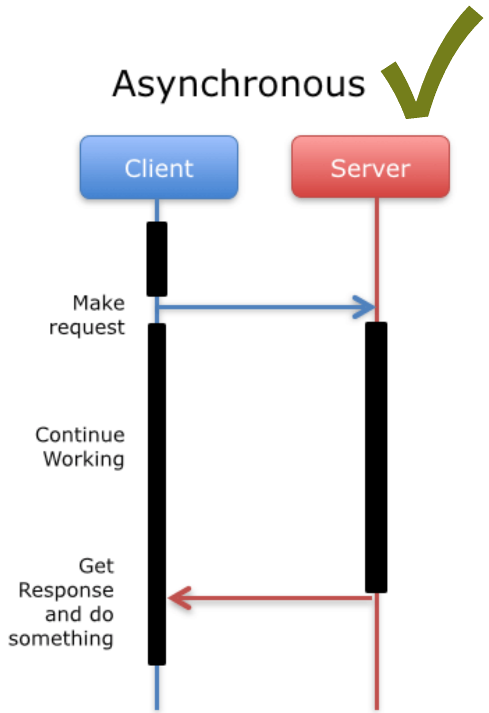
Transmission Control Protocol (TCP)


Active Opener (Client)
Passive Opener (Server)
SYN, Seq = ISN(c), (options)
SYN + ACK, Seq = ISN(s), ACK = ISN(c) + 1, (options)
ACK, Seq = ISN(c) + 1, ACK = ISN(s) + 1, (options)
Connection Set-Up (Three-Way Handshake)
Data Transfer (Established)
[No Data Transferred in This Example]
FIN+ACK, Seq = K, ACK = L, (options)
ACK, Seq = L, ACK = K + 1, (options)
FIN + ACK, Seq = L, ACK = K + 1, (options)
ACK, Seq = K, ACK = L + 1, (options)
Connection Close (Modified Three-Way Handshake)

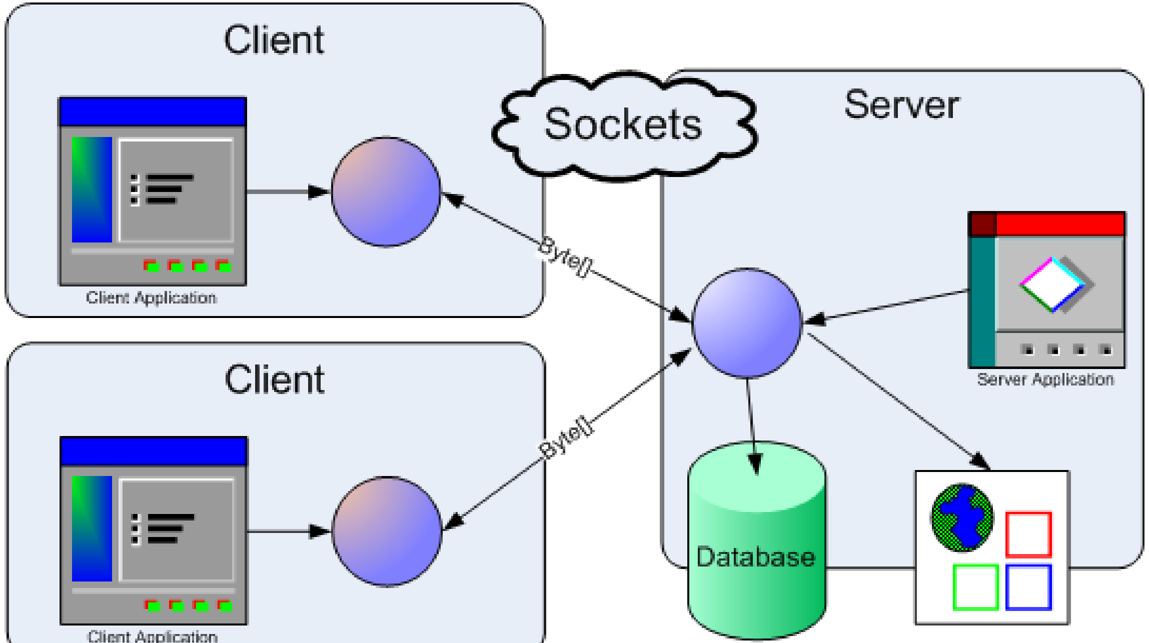
TCP SOCKET PROGRAMMING IN NODE.JS

Node.js “net” module





Node.js “net” module



Simple TCP server

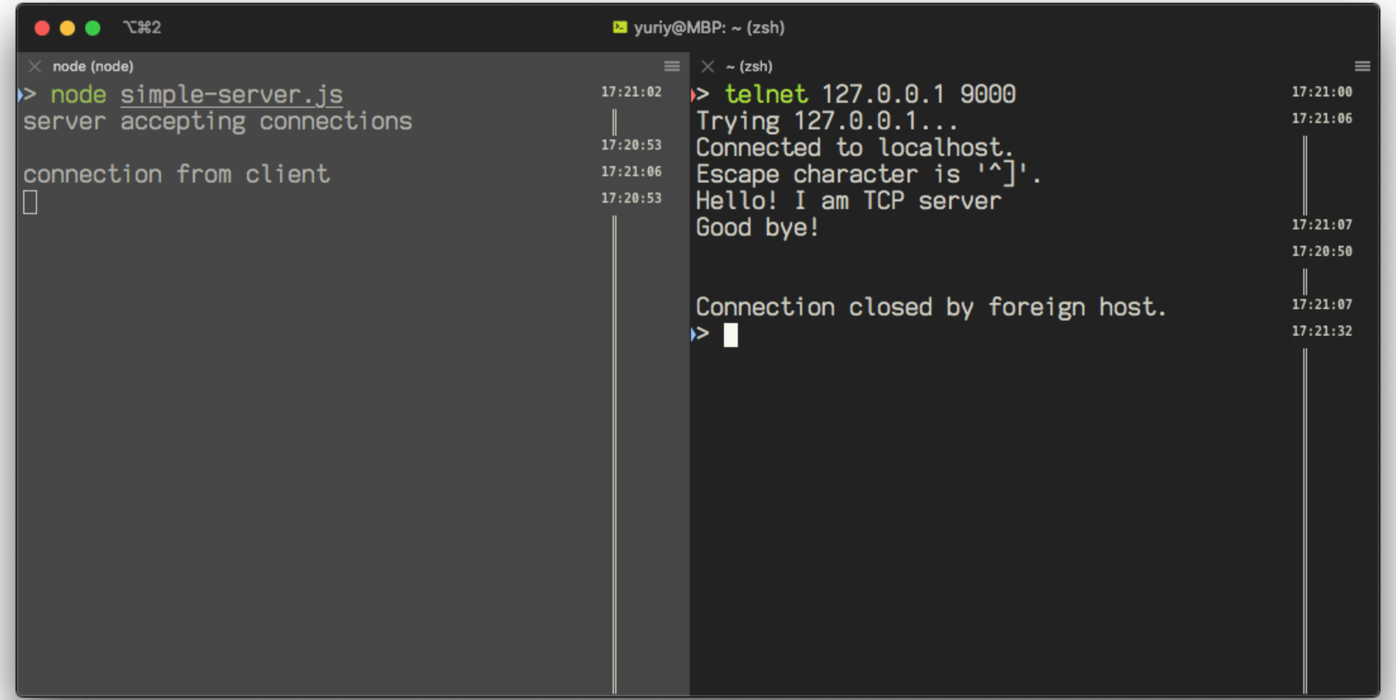
net.createServer([options][, connectionlistener])

- options
  - allowHalfOpen – Indicates whether half-opened TCP connections are allowed. Default: false.
  - pauseOnConnect – Indicates whether the socket should be paused on incoming connections. Default: false.
- connectionListener – Automatically set as a listener for the 'connection' event.
- Returns: <net.Server>

```
function createServer(
    options?: {
        allowHalfOpen?: boolean,
        pauseOnConnect?: boolean
    },
    connectionListener?: (socket: Socket) => void
): Server;
```

```
const net = require('net');

const server = net.createServer({}, (tcpSocket) => {
    console.log('connection from client');
    tcpSocket.write('Hello! I am TCP server\n');

    setTimeout(() => {
        tcpSocket.end('Good bye!\n\n\n');
    }, 1000);
});

server.listen(9000, 'localhost', 2);

server.on('listening', () => {
    console.log('server accepting connections\n');
});
```

Simple TCP server



```
⌥⌘2                          yuriy@MBP: ~ (zsh)

node (node)
> node simple-server.js                          17:21:02
server accepting connections
                                                 17:20:53
connection from client                           17:21:06
                                                 17:20:53
```

```
~ (zsh)
> telnet 127.0.0.1 9000                          17:21:00
Trying 127.0.0.1...                              17:21:06
Connected to localhost.
Escape character is '^]'.
Hello! I am TCP server
Good bye!                                        17:21:07
                                                 17:20:50

Connection closed by foreign host.               17:21:07
>                                                17:21:32
```

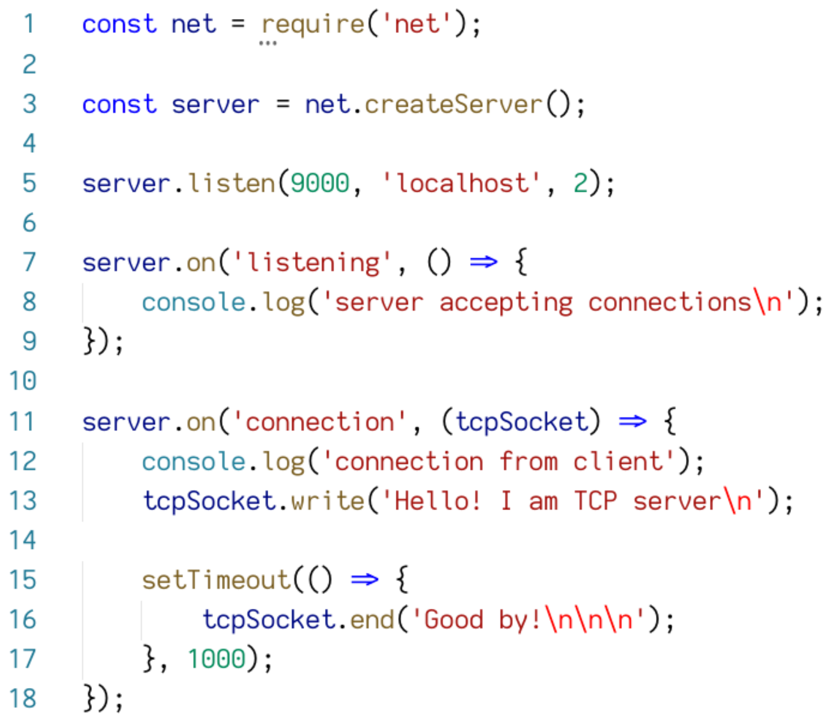
Simple TCP server with ‘connection’ event listener

For TCP servers:

- server.listen([port][, host][, backlog][, callback])

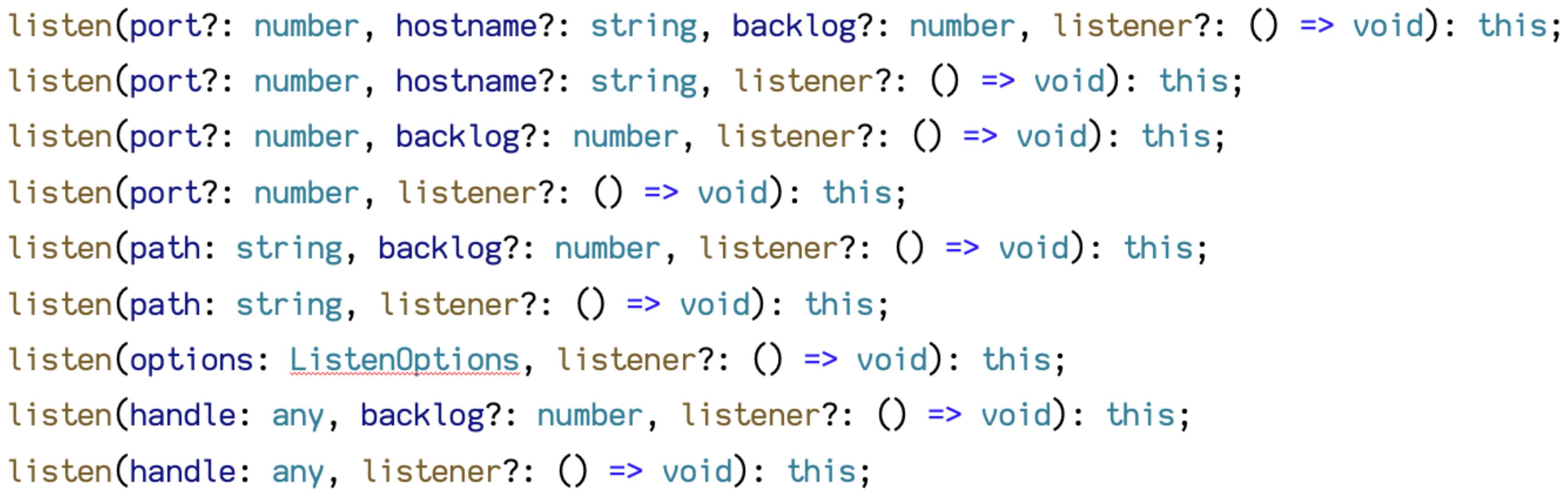
For IPC servers:

- server.listen(path[, backlog][, callback])



```
const net = require('net');

const server = net.createServer();

server.listen(9000, 'localhost', 2);

server.on('listening', () => {
    console.log('server accepting connections\n');
});

server.on('connection', (tcpSocket) => {
    console.log('connection from client');
    tcpSocket.write('Hello! I am TCP server\n');

    setTimeout(() => {
        tcpSocket.end('Good by!\n\n\n');
    }, 1000);
});
```

Server.listen() overloaded signatures



```
listen(port?: number, hostname?: string, backlog?: number, listener?: () => void): this;
listen(port?: number, hostname?: string, listener?: () => void): this;
listen(port?: number, backlog?: number, listener?: () => void): this;
listen(port?: number, listener?: () => void): this;
listen(path: string, backlog?: number, listener?: () => void): this;
listen(path: string, listener?: () => void): this;
listen(options: ListenOptions, listener?: () => void): this;
listen(handle: any, backlog?: number, listener?: () => void): this;
listen(handle: any, listener?: () => void): this;
```

Simple TCP echo server

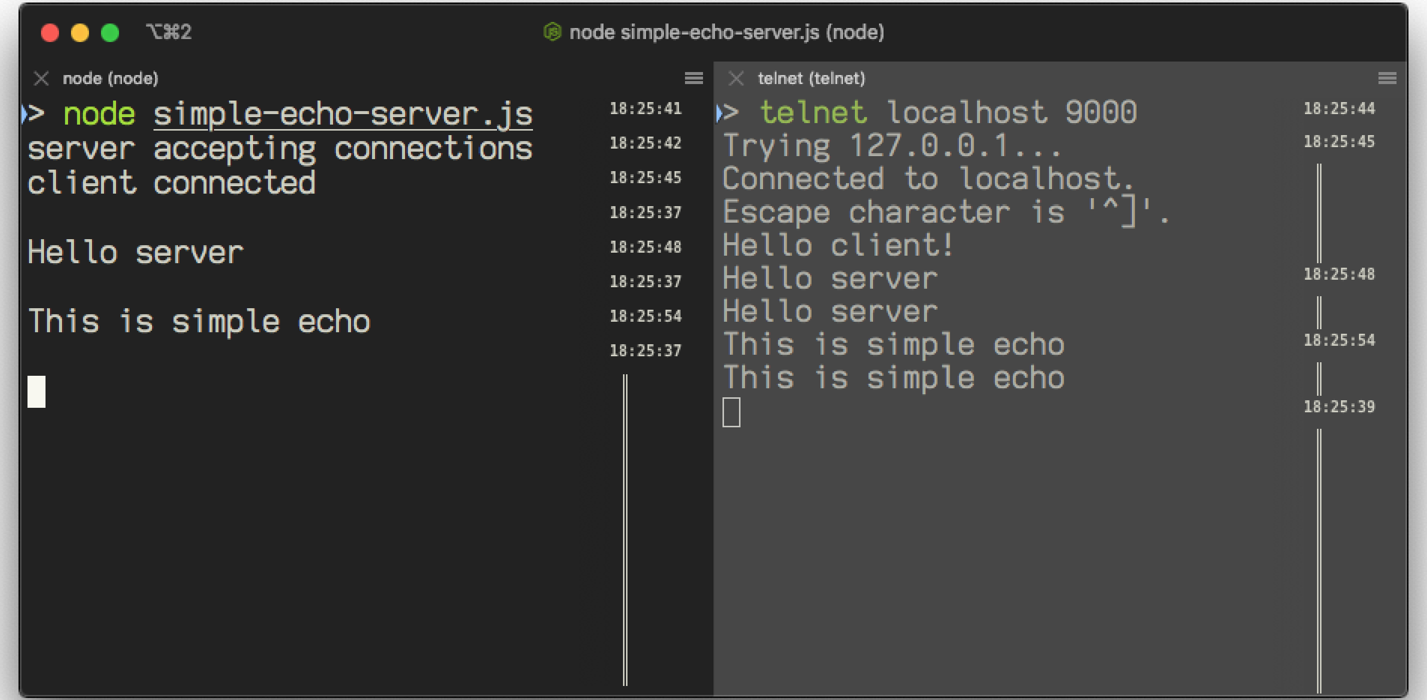
```
class Server extends EventEmitter {
  on(
    event: "connection",
    listener: (socket: Socket) => void,
  ): this;
}
```

'socket' object is a duplex stream:

- socket.write(data[, encoding, callback])
- socket.pipe(destination[, options])

```
const net = require('net');
const server = net.createServer();

server.listen(9000, 'localhost', 2);
server.on('listening', function () {
    console.log('server accepting connections');
});

server.on('connection', (tcpSocket) => {
    console.log('client connected\n');
    tcpSocket.write('Hello client!\n');

    tcpSocket.on('data', (data) => {
        console.log(data.toString());
    });

    tcpSocket.pipe(tcpSocket);
});
```

Simple TCP echo server



```
> node simple-echo-server.js                 18:25:41
server accepting connections                 18:25:42
client connected                             18:25:45
                                             18:25:37
Hello server                                 18:25:48
                                             18:25:37
This is simple echo                          18:25:54
                                             18:25:37
```

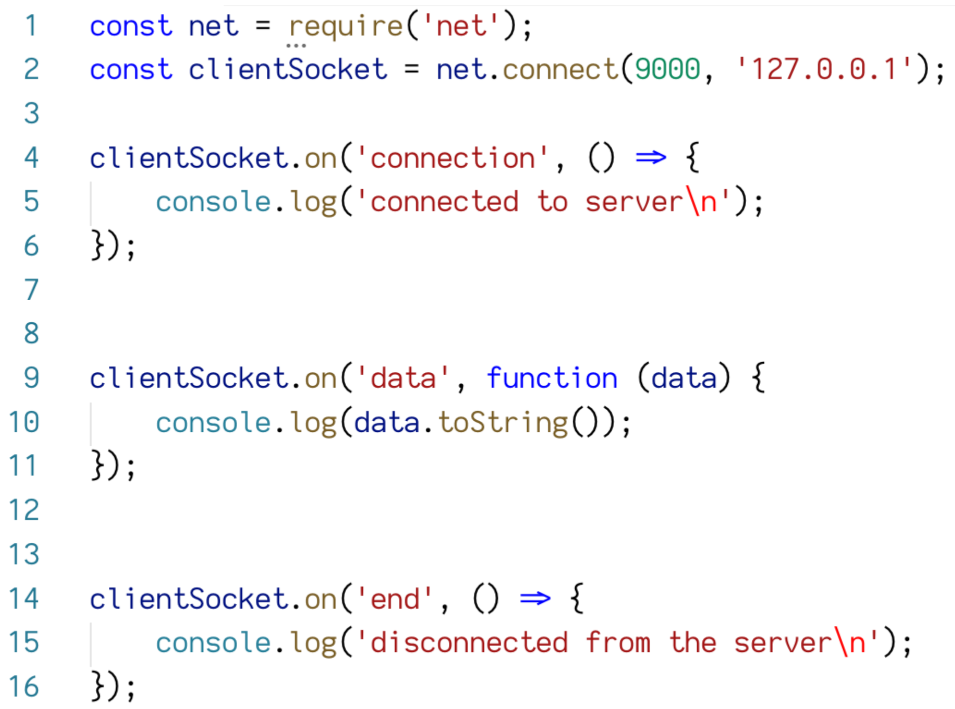
```
> telnet localhost 9000                      18:25:44
Trying 127.0.0.1...                          18:25:45
Connected to localhost.
Escape character is '^]'.
Hello client!
Hello server                                 18:25:48
Hello server
This is simple echo                          18:25:54
This is simple echo
                                             18:25:39
```

Simple TCP client

net.connect() aliases to net.createConnection().

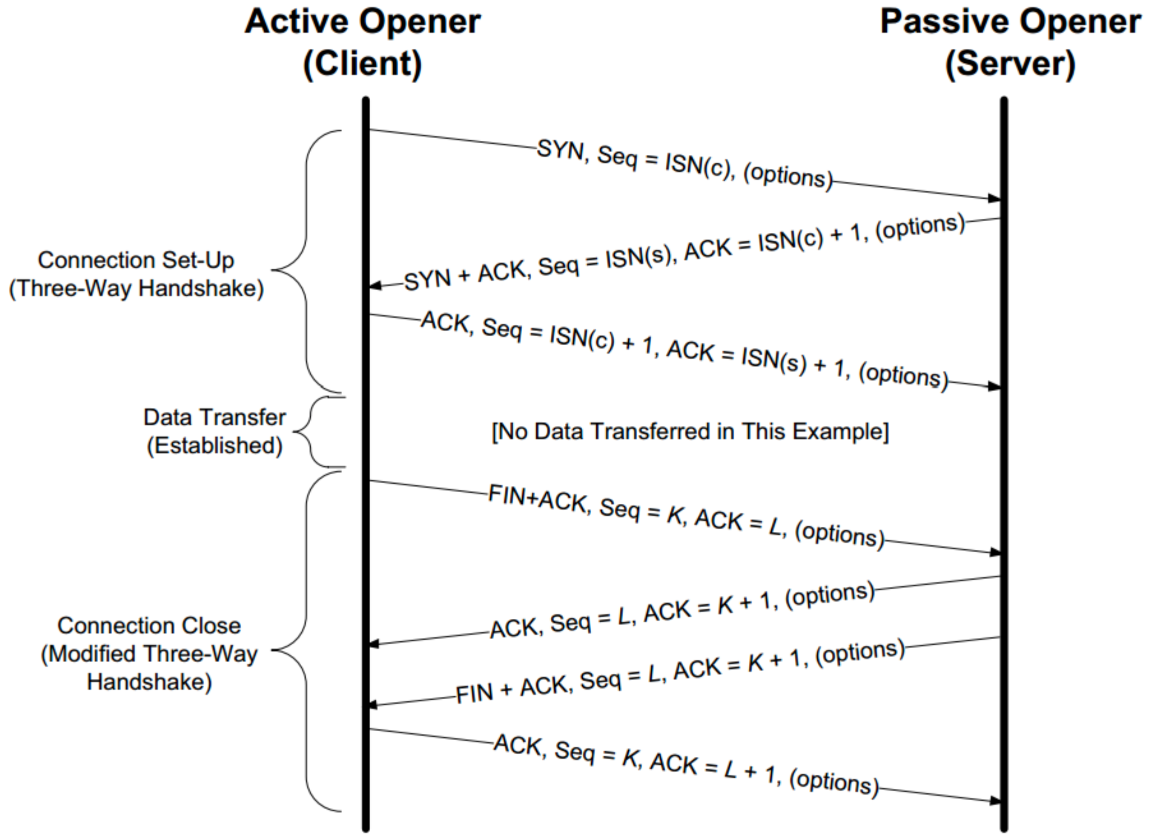
net.connect(port[, host][, connectListener]) for TCP connections.

net.connect(path[, connectListener]) for IPC connections.



```
const net = require('net');
const clientSocket = net.connect(9000, '127.0.0.1');

clientSocket.on('connection', () => {
    console.log('connected to server\n');
});


clientSocket.on('data', function (data) {
    console.log(data.toString());
});


clientSocket.on('end', () => {
    console.log('disconnected from the server\n');
});
```

Transmission Control Protocol (TCP)


Active Opener (Client)
Passive Opener (Server)
Connection Set-Up (Three-Way Handshake)
SYN, Seq = ISN(c), (options)
SYN + ACK, Seq = ISN(s), ACK = ISN(c) + 1, (options)
ACK, Seq = ISN(c) + 1, ACK = ISN(s) + 1, (options)
Data Transfer (Established)
[No Data Transferred in This Example]
Connection Close (Modified Three-Way Handshake)
FIN+ACK, Seq = K, ACK = L, (options)
ACK, Seq = L, ACK = K + 1, (options)
FIN + ACK, Seq = L, ACK = K + 1, (options)
ACK, Seq = K, ACK = L + 1, (options)

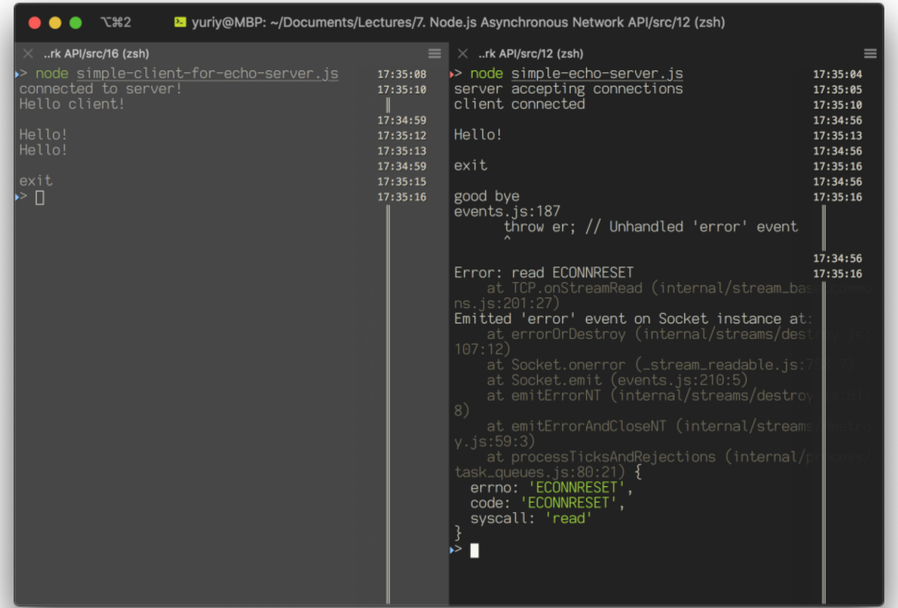
Simple TCP client (echo server)

socket.end(‘message’) is equivalent to calling socket.write(‘message’, ‘encoding’) followed by socket.end().

```
end(cb?: () => void): void;
end(buffer: Uint8Array | string, cb?: () => void): void;
end(str: Uint8Array | string, encoding?: string, cb?: () => void): void;
```

```
const net = require('net');
const socket = net.connect({ port: 9000 });

socket.on('connect', () => {
    console.log('connected to server!');
});

socket.on('data', (data) => {
    console.log(data.toString());
});

process.stdin.pipe(socket);
process.stdin.on('data', (data) => {
    const str = data.toString();
    if (str.trim() === 'exit') {
        socket.end('good bye');
        process.exit();
    }
});

process.on('SIGINT', () => {
    console.log('Caught interrupt signal');
    if (socket) {
        socket.end('terminated');
        process.exit();
    }
});
```

Simple TCP client (echo server)



```
⌥⌘2    yuriy@MBP: ~/Documents/Lectures/7. Node.js Asynchronous Network API/src/12 (zsh)
```

```
..rk API/src/16 (zsh)
> node simple-client-for-echo-server.js       17:35:08
connected to server!                          17:35:10
Hello client!
                                              17:34:59
Hello!                                        17:35:12
Hello!                                        17:35:13
                                              17:34:59
exit                                          17:35:15
>                                             17:35:16
```

```
..rk API/src/12 (zsh)
> node simple-echo-server.js                  17:35:04
server accepting connections                  17:35:05
client connected                              17:35:10
                                              17:34:56
Hello!                                        17:35:13
                                              17:34:56
exit                                          17:35:16
                                              17:34:56
good bye                                      17:35:16
events.js:187
      throw er; // Unhandled 'error' event
      ^
                                              17:34:56
Error: read ECONNRESET                        17:35:16
    at TCP.onStreamRead (internal/stream_base_commons.js:201:27)
Emitted 'error' event on Socket instance at:
    at errorOrDestroy (internal/streams/destroy.js:107:12)
    at Socket.onerror (_stream_readable.js:7[illegible]:7)
    at Socket.emit (events.js:210:5)
    at emitErrorNT (internal/streams/destroy.js:91:8)
    at emitErrorAndCloseNT (internal/streams/destroy.js:59:3)
    at processTicksAndRejections (internal/process/task_queues.js:80:21) {
  errno: 'ECONNRESET',
  code: 'ECONNRESET',
  syscall: 'read'
}
>
```

```javascript
const net = require('net');
const socket = net.connect({ port: 9000 });

socket.on('connect', () => {
    console.log('connected to server!');
});

socket.on('data', (data) => {
    console.log(data.toString());
});

process.stdin.pipe(socket);
process.stdin.on('data', (data) => {
    const str = data.toString();
    if (str.trim() === 'exit') {
        socket.end('good bye');
        process.exit();
    }
});

process.on('SIGINT', () => {
    console.log('Caught interrupt signal');
    if (socket) {
        socket.end('terminated');
        process.exit();
    }
});
```

Error handling

```javascript
const net = require('net');
const server = net.createServer();

server.listen(9000, 'localhost', 2);
server.on('listening', function () {
    console.log('server accepting connections');
});

server.on('connection', (tcpSocket) => {
    console.log('client connected\n');
    tcpSocket.write('Hello client!\n');

    tcpSocket.on('data', (data) => {
        console.log(data.toString());
    });

    tcpSocket.on('error', (error) => {
        console.log('Connection error: ', error.stack);
    });

    tcpSocket.on('end', () => {
        console.log('FIN frame received');
    });

    tcpSocket.on('close', () => {
        console.log('Connection ended');
    });

    tcpSocket.pipe(tcpSocket);
});
```

```javascript
const net = require('net');
const socket = net.connect({ port: 9000 });

socket.on('connect', () => {
    console.log('connected to server!');
});

socket.on('data', (data) => {
    console.log(data.toString());
});

socket.on('close', (hadError) => {
    console.log(hadError);
    process.exit();
});

process.stdin.pipe(socket);

process.stdin.on('data', (data) => {
    const str = data.toString();
    if (str.trim() === 'exit') {
        socket.end('good bye');
    }
});

process.on('SIGINT', () => {
    console.log('Caught interrupt signal');
    if (socket) {
        socket.end('terminated');
    }
});
```

IPC SUPPORT IN NODE.JS

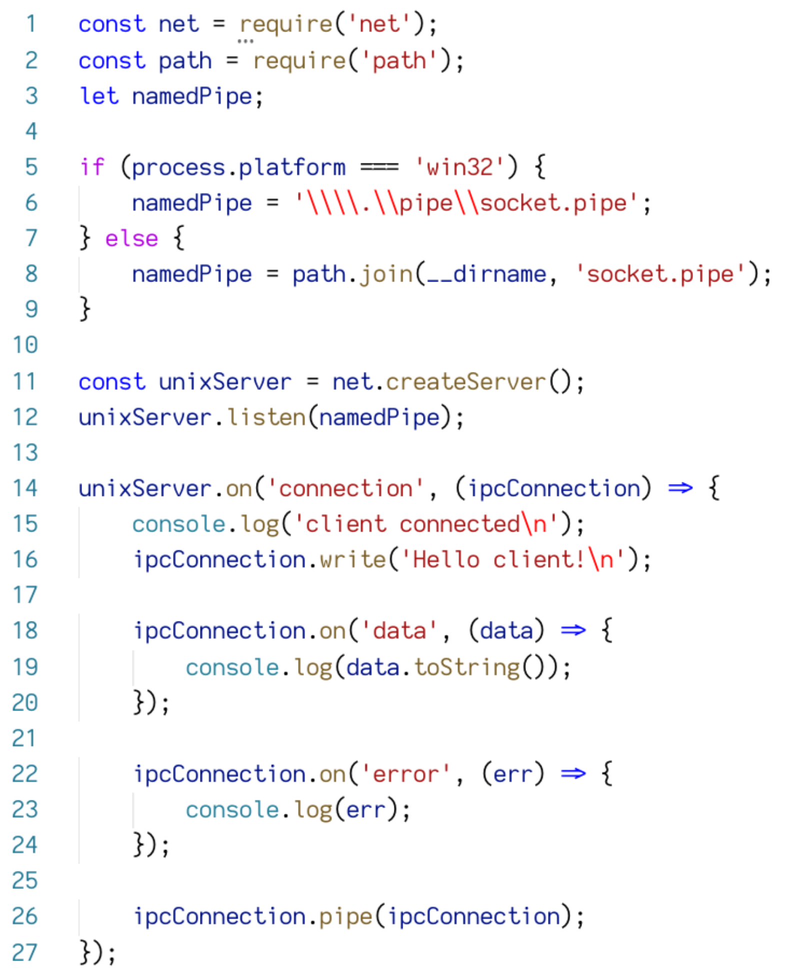
Named pipes (server)

server.listen(path[, backlog][, callback])

- Start an IPC server listening for connections on the given path.

server.listen([port][, host][, backlog][, callback])

- Start a TCP server listening for connections on the given port and host.



```javascript
const net = require('net');
const path = require('path');
let namedPipe;

if (process.platform === 'win32') {
    namedPipe = '\\\\.\\pipe\\socket.pipe';
} else {
    namedPipe = path.join(__dirname, 'socket.pipe');
}

const unixServer = net.createServer();
unixServer.listen(namedPipe);

unixServer.on('connection', (ipcConnection) => {
    console.log('client connected\n');
    ipcConnection.write('Hello client!\n');

    ipcConnection.on('data', (data) => {
        console.log(data.toString());
    });

    ipcConnection.on('error', (err) => {
        console.log(err);
    });

    ipcConnection.pipe(ipcConnection);
});
```

Named pipes (client)

net.connect(path[, connectListener])

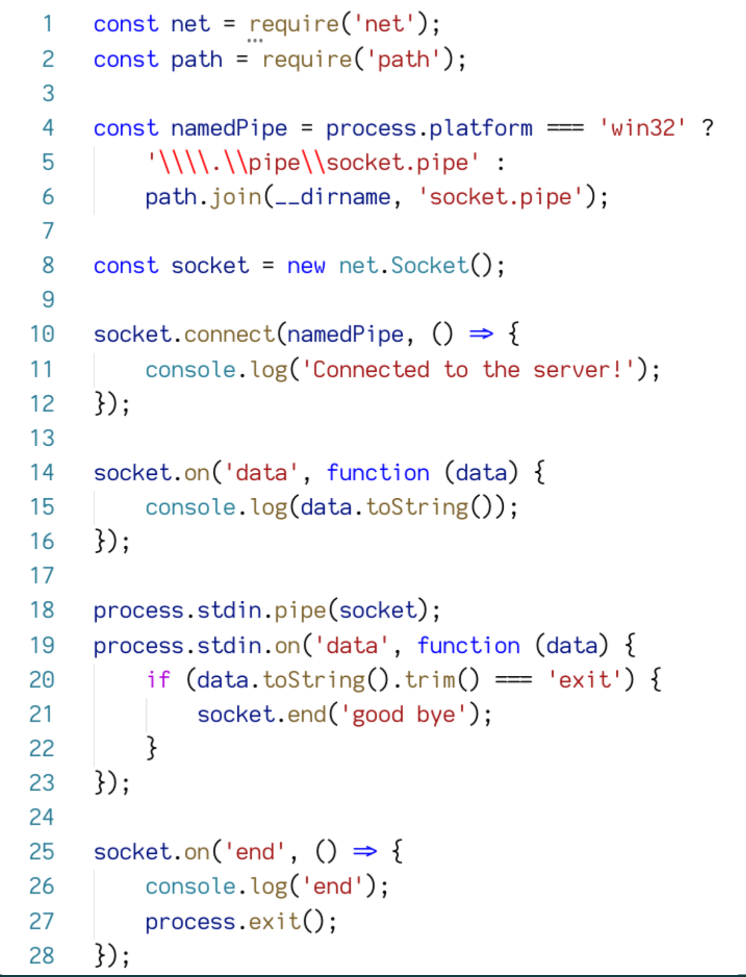
net.createConnection(path[, connectListener])

- Initiates an IPC connection

net.connect(port[, host][, connectListener])

net.createConnection(port[,host],connectListener])

- Initiates a TCP connection



```javascript
const net = require('net');
const path = require('path');

const namedPipe = process.platform === 'win32' ?
    '\\\\.\\pipe\\socket.pipe' :
    path.join(__dirname, 'socket.pipe');

const socket = new net.Socket();

socket.connect(namedPipe, () => {
    console.log('Connected to the server!');
});

socket.on('data', function (data) {
    console.log(data.toString());
});

process.stdin.pipe(socket);
process.stdin.on('data', function (data) {
    if (data.toString().trim() === 'exit') {
        socket.end('good bye');
    }
});

socket.on('end', () => {
    console.log('end');
    process.exit();
});
```

Useful ‘Socket’ object methods

```
interface AddressInfo {
  address: string;
  family: string;
  port: number;
}

class Socket extends stream.Duplex {
  setEncoding(encoding?: string): this;
  pause(): this;
  resume(): this;
  setTimeout(timeout: number, callback?: () => void): this;
  setNoDelay(noDelay?: boolean): this;
  setKeepAlive(enable?: boolean, initialDelay?: number): this;
  address(): AddressInfo | string;

  readonly bufferSize: number;
  readonly bytesRead: number;
  readonly bytesWritten: number;
}
```

CONFIDENTIAL | © 2019 EPAM Systems, Inc.

Useful links

- https://nodejs.org/api/index.html – the official ‘net’ module documentation
- https://nodejs.org/api/dgram.html – the ‘dgram’ module for UDP Datagram sockets
- https://github.com/RIAEvangelist/node-ipc – ‘node-ipc’ module for fast inter-process communication
- https://docs.microsoft.com/en-us/windows/win32/ipc/pipe-names – Windows pipe names
- https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi – overview of OSI model from CloudFlare