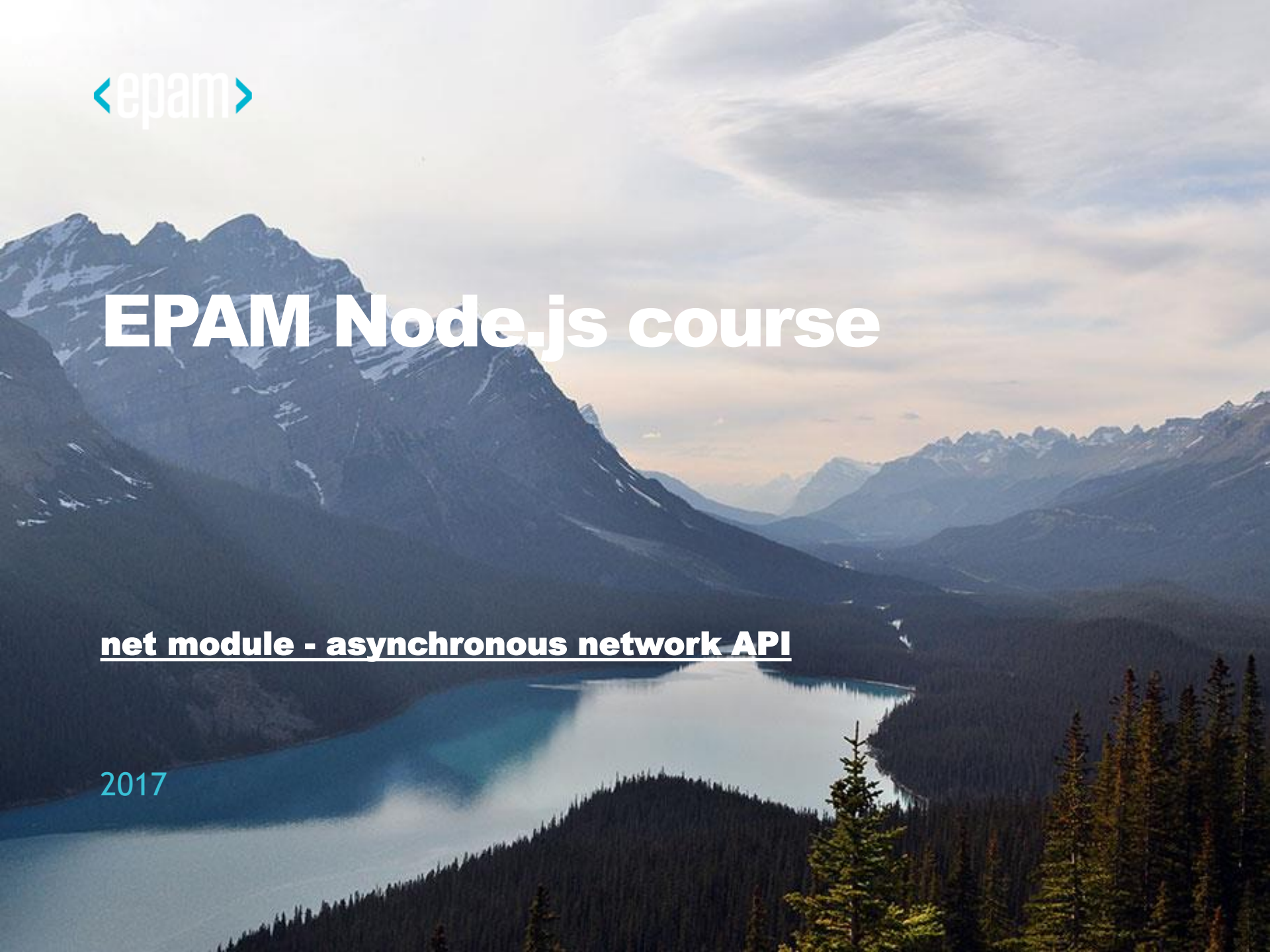


EPAM Node.js course

net module - asynchronous network API

2017

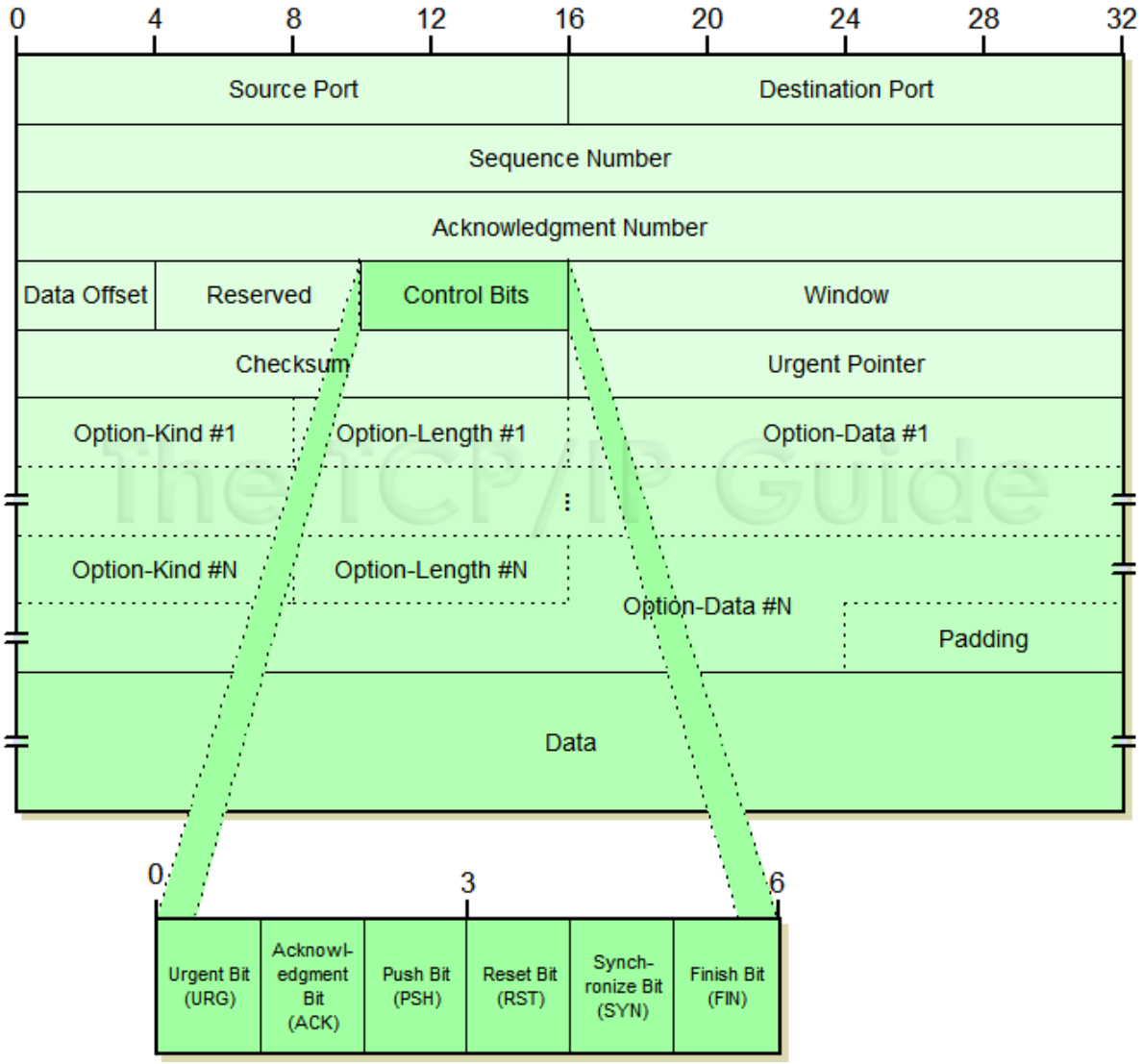
Agenda

1. OSI Model, Transmission Control Protocol (TCP)
2. Standard Node.js NET module overview
3. Simple TCP server and client
4. TCP echo server and client
5. Error handling
6. IPC (interprocess communication) in NET module
7. Additional information

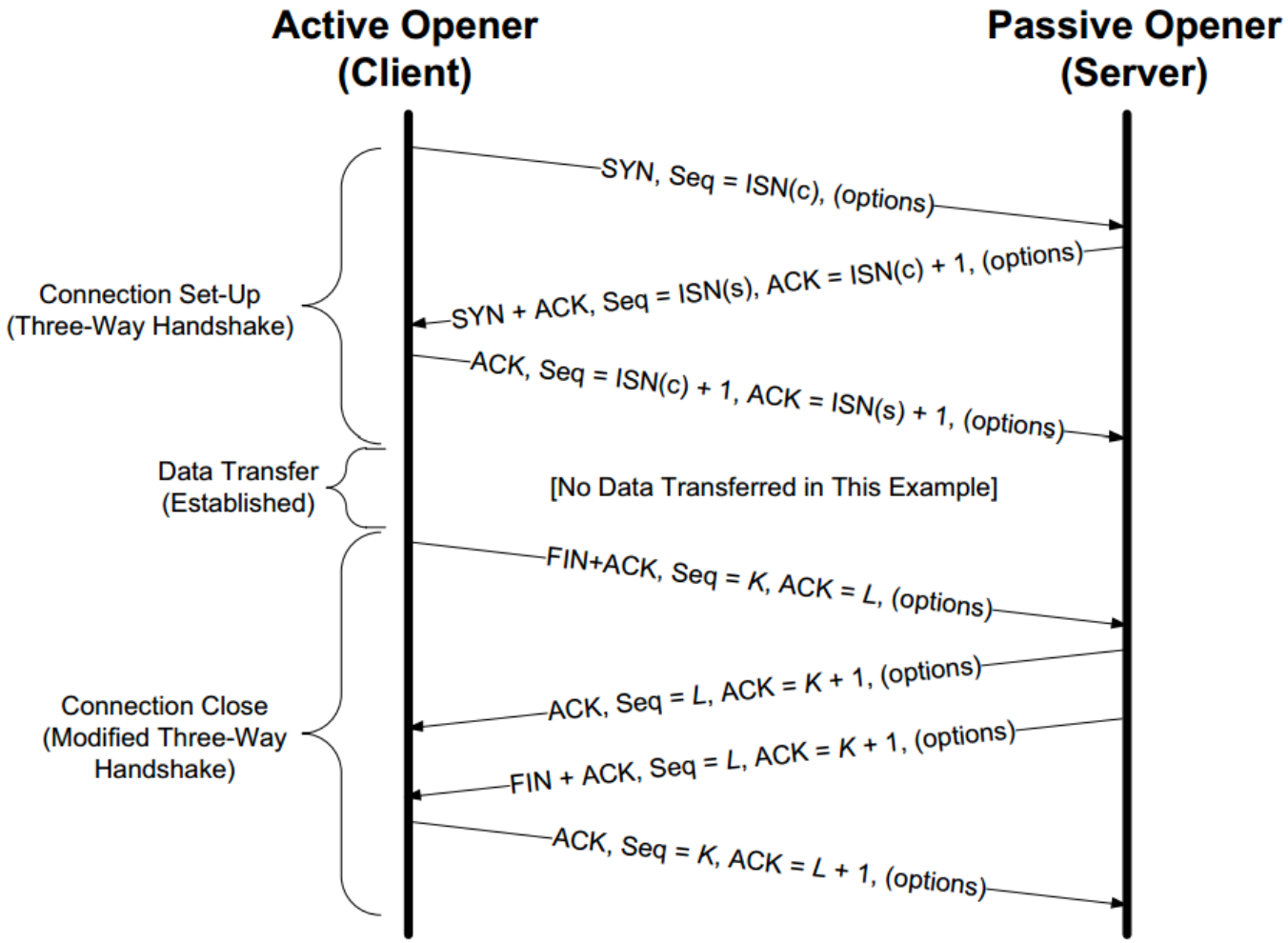
Open Systems Interconnection model (OSI Model)

Layer	Function	Example
Application (7)	Services that are used with end user applications	SMTP,
Presentation (6)	Formats the data so that it can be viewed by the user Encrypt and decrypt	JPG, GIF, HTTPS, SSL, TLS
Session (5)	Establishes/ends connections between two hosts	NetBIOS, PPTP
Transport (4)	Responsible for the transport protocol and error handling	TCP, UDP
Network (3)	Reads the IP address form the packet.	Routers, Layer 3 Switches
Data Link (2)	Reads the MAC address from the data packet	Switches
Physical (1)	Send data on to the physical wire.	Hubs, NICS, Cable

Transmission Control Protocol (TCP)



Transmission Control Protocol (TCP)



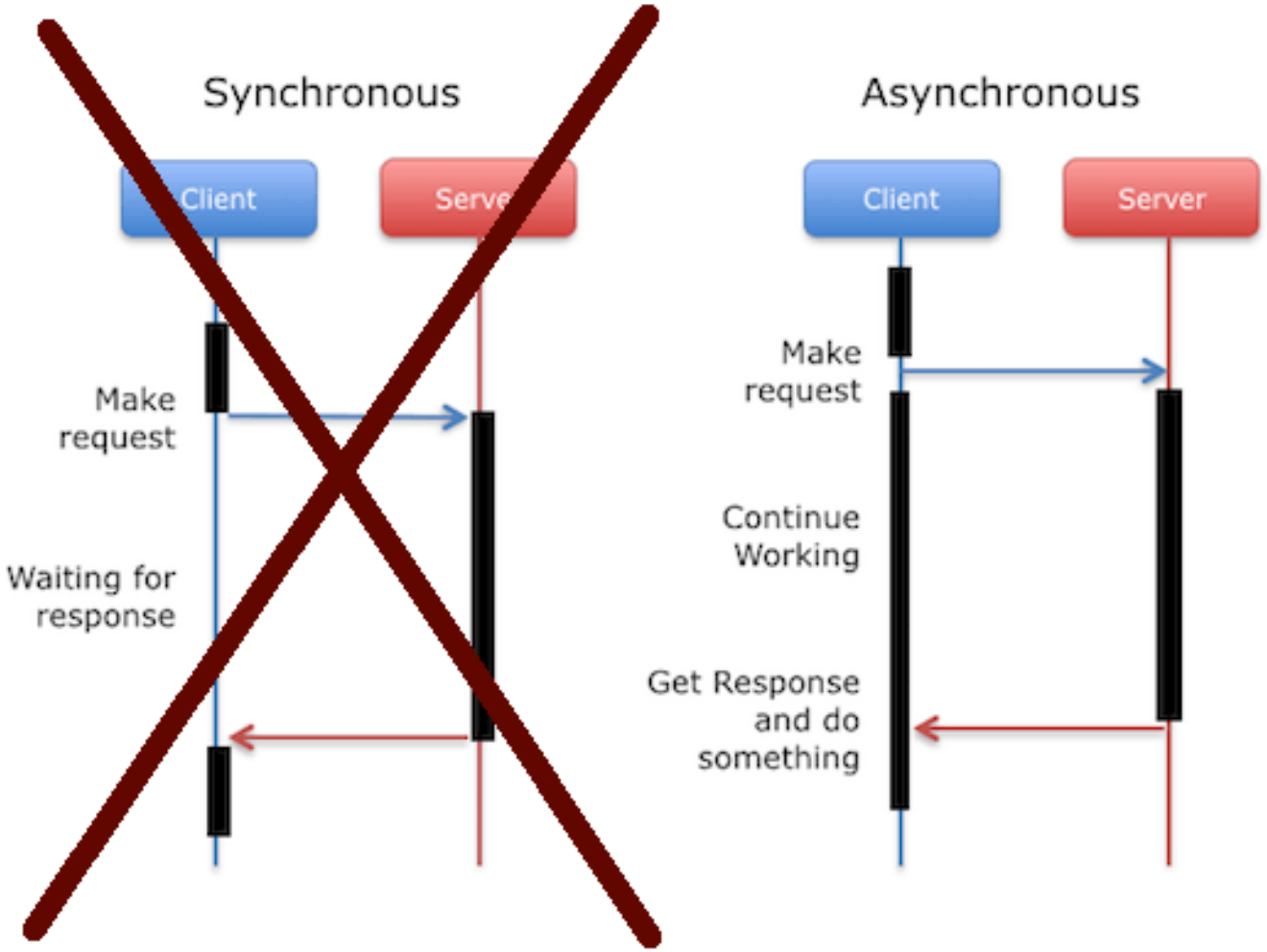
<epam>

TCP Socket Programming in Node.js

NET MODULE

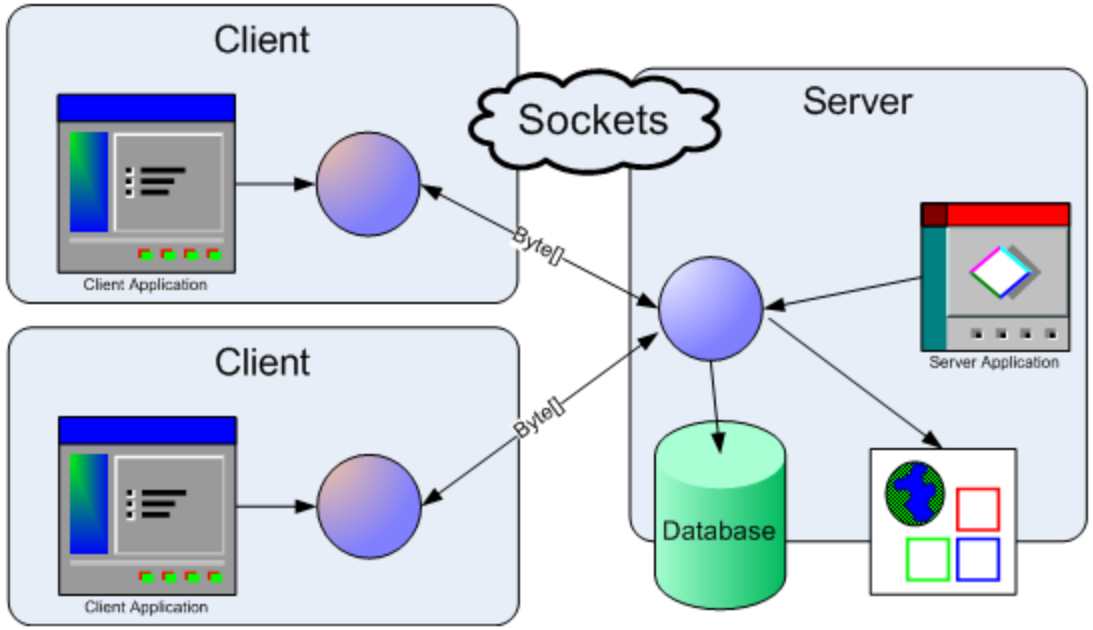
2017

Node.js net module



The net module provides asynchronous network API

NET MODULE


Client
Client Application
Client
Client Application
Sockets
Byte[]
Byte[]
Server
Server Application
Database

Simple TCP server

```
1  const net = require('net');
2  const server = net.createServer({}, (socket) => {
3    console.log('connection from client');
4    socket.write('Hello! I am TCP server\n');
5
6    setTimeout(() => {
7       socket.end('Good by!\n\n\n');
8    }, 1000);
9  });
10
11 server.listen(9000, 'localhost', 2);
12
13 server.on('listening', () => {
14    console.log('server accepting connections\n');
15 });
```

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

options: {

- allowHalfOpen: Boolean,
- pauseOnConnect: Boolean

– }

connListener — Function

```
[vadim@clearwater slide-7]$ node simple-server.js
server accepting connections
```

```
[vadim@clearwater ~]$ telnet 127.0.0.1 9000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Hello! I am TCP server
Good by!


Connection closed by foreign host.
[vadim@clearwater ~]$
```

Simple TCP server (with ‘connection’ event listener)

```
1  const net = require('net');
2
3  const server = net.createServer();
4
5  server.listen(9000, 'localhost', 2);
6
7  server.on('listening', () => {
8     console.log('server accepting connections\n');
9  });
10
11 server.on('connection', (tcpSocket) => {
12    console.log('connection from client');
13    tcpSocket.write('Hello! I am TCP server\n');
14
15    setTimeout(() => {
16       tcpSocket.end('Good by!\n\n\n');
17    }, 1000);
18 });
```

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

server.on('connection', callback);

Simple TCP server (with ‘connection’ event listener)

```
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);
});
```

For TCP servers:

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

For IPC servers:

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

Simple TCP echo server

```
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', (socket) => {
    console.log('client connected\n');
    socket.write('Hello client!\n');


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

socket instanceof stream.Duplex === true;

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

```
[vadim@clearwater slide-10]$ node simple-echo-server.js
server accepting connections
client connected

Hello there!

is this a simple echo server?

^C
[vadim@clearwater slide-10]$
```

```
[vadim@clearwater ~]$ telnet 127.0.0.1 9000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Hello client!
Hello there!
Hello there!
is this a simple echo server?
is this a simple echo server?
Connection closed by foreign host.
[vadim@clearwater ~]$
```

Simple TCP client

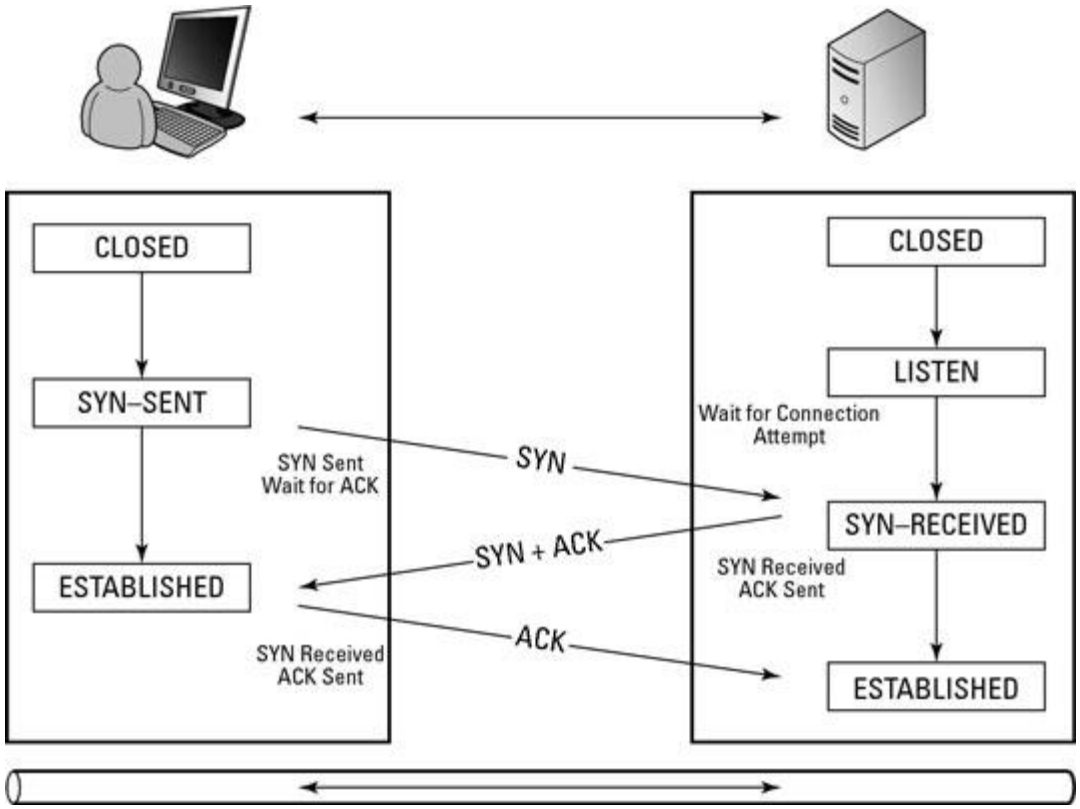
```
const net = require('net');
const clientSocket = net.connect(9000, localhost);


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


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


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

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

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

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

Transmission Control Protocol (TCP)



Simple TCP client (echo server)

```
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 by');
        process.exit();
    }
});


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

socket.end([data][, encoding]);

socket.end('good by') is equivalent to calling:
socket.write(data, encoding) followed by socket.end()

Simple TCP client (echo server)

```
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 by');
        process.exit();
    }
});


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

```
[vadim@clearwater slide-12]$ node simple-client-for-echo-server.js
connected to server!
Hello client!

Hi
Hi

exit
[vadim@clearwater slide-12]$
```

```
[vadim@clearwater slide-13]$ node simple-echo-server.js
server accepting connections
client connected

Hi

exit

good by
events.js:182
      throw er; // Unhandled 'error' event
      ^

Error: read ECONNRESET
    at exports._errnoException (util.js:1024:11)
    at TCP.onread (net.js:610:25)
[vadim@clearwater slide-13]$
```

Error handling

```
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) => {
    process.exit();
});


process.stdin.pipe(socket);


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


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

```
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);
});
```

<epam>

IPC Support in Net module

2017

UNIX domain socket / Named pipes (server)

```
1  const net = require('net');
2  const path = require('path');
3    let namedPipe;
4
5    if (process.platform === 'win32') {
6      namedPipe = '\\\\.\\pipe\\pipe.name';
7    } else {
8      namedPipe = path.join(__dirname,'pipe.name');
9    }
10
11   const unixServer = net.createServer();
12   unixServer.listen(namedPipe);
13
14   unixServer.on('connection', (ipcConnection) => {
15     console.log('client connected\n');
16     ipcConnection.write('Hello client!\n');
17
18     ipcConnection.on('data', (data) => {
19         console.log(data.toString());
20       });
21
22   ipcConnection.on('error', function(err) {
23       console.log(err);
24   });
25
26   ipcConnection.pipe(ipcConnection);
27  });
```

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

UNIX domain socket / Named pipes (client)

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


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


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 by');
    }
});


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

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

Usuful methods

- **socket.address() – returns @object e.g. { port: 12346, family: 'IPv4', address: '127.0.0.1' }**
- **socket.setKeepAlive([enable][, initialDelay]) — returns the socket itself.**
- **socket.setTimeout(timeout[, callback]) — returns the socket itself.**
  - `socket.setTimeout(3000);`
  - `socket.on('timeout', () => {`
  - `    console.log('socket timeout');`
  - `    socket.end();`
  - `});`
- **socket.setEncoding([encoding]) — returns the socket itself.**
- **socket.bytesWritten**
- **socket.bytesRead**

Useful links

https://nodejs.org/dist/latest-v8.x/docs/api/net.html - The net module documentation

https://nodejs.org/api/dgram.html - The dgram module provides an implementation of UDP Datagram sockets.

https://www.npmjs.com/package/node-ipc - node-ipc module

https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa365783(v=vs.85).aspx - Pipe Names

https://support.microsoft.com/en-in/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained - The OSI Model's Seven Layers Defined and Functions Explained