TCP

TCP is a transport protocol that is:

- connection-oriented
- ensures data arrives in order
- endures data arrives undamaged

Many protocols are built on top of TCP, like SMTP and HTTP.

In Node, the `net` module has a server and client implementations.

TELNET

Extremely basic tcp client

00_telnet.js setup

```
if (process.argv < 3) { return console.log('Usage: node 00_telnet.js hos

var net = require('net');
var readline = require('readline');
var stream = require('stream');
var host = process.argv[2].split(':');
var cli = readline.createInterface(process.stdin, process.stdout);

var socket = net.connect({ host : host[0], port : host[1] }, function()

  console.log('connected to', host[0], 'on port', host[1]);

  /// Bind socket to readline and vice versa

});

socket.on('error', function(err) {
  console.log('ERROR:', err.message);
});
```

00_telnet.js bind socket to readline

```
/// Bind socket to readline and vice versa
cli.setPrompt('$ ');
cli.prompt();
cli.on('line', function(line) {
 socket.write(line + '\r\n');
});

var transformer = new stream.Transform();
transformer._transform = function(buf, encoding, done) {
  this.push(buf);
  cli.prompt();
  done();
};

socket.pipe(transformer).pipe(process.stdout);

socket.on('end', function() {
  console.log('disconnected from server');
  process.exit();
});
```

```
$ node 00_telnet.js localhost:8000
```

TCP SERVER

01_server.js:

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

server.on('connection', function(socket) {
  console.log('new connection from', socket.remoteAddress);

  socket.setEncoding('utf8');

  socket.on('readable', function() {
    var data;
    while(data = socket.read()) {
      console.log(socket.remoteAddress + ':' + socket.remotePort + '>',
    }
  });

  socket.once('end', function() {
    console.log(socket.remoteAddress, 'disconnected');
  });
}).listen(8000, function() {
  console.log('TCP server is listening on port 8000');
});
```

STREAMING

Use a `Transform` to output client messages to `stdout`

02_server_stream.js create a transform

```
/// Add a transform stream
function createRemoteInfoTransform(socket) {
  var out = new Transform({ decodeStrings: 'utf8' });

  out.prefix = socket.remoteAddress + ':' + socket.remotePort;

  out._transform = function(buf, encoding, done) {
    this.push(out.prefix + ' > ' + buf);
    done();
  };

  return out;
}
```

02_server_stream.js hook it up

```
/// Use socket.pipe
  var transformer = createRemoteInfoTransform(socket);

  socket.pipe(transformer).pipe(process.stdout);
```

CHATROOM

03_chat_streams.js setup a chat room

```
/// Create a room
var room = new stream.PassThrough();
```

03_chat_streams.js hook it up

```
/// Plumbing
  var transformer = createRemoteInfoTransform(socket);

  socket.on('error', function() {
    console.log(transformer.prefix, 'has timed out');
  });


  socket.pipe(transformer);

  // Split
  transformer.pipe(process.stdout);
  transformer.pipe(room, { end: false }).pipe(socket);
```

Here we're using a pass-through stream as a chat room to echo all incoming data to all the connections.

TCP CLIENT

TCP client that connects to a chat server and sends a random phrase:

04_tcp_client_random.js client

```
var port = parseInt(process.argv[2], 10) || 8000;
var host = process.argv[3] || 'localhost'

var net = require('net');
var client = net.connect(port, host);
client.setEncoding('utf8');

client.once('connect', function() {
  setInterval(function() {
    client.write(randomPhrase() + '\n');
  }, 5000);
});

/// Random Phrase
```

Random phrase generation:

04_tcp_client_random.js

```
/// Random Phrase
var sets = [
  ['the', 'many', 'some']
, ['repeating', 'twelve', 'immortal', 'sequel']
, ['monkey', 'cat', 'dog', 'shark']
, ['sighed', 'cried', 'ran', 'barked', 'purred', 'swam']
, ['fast', 'slow', 'carefully', 'perfectly']
];

function randomPhrase() {
  var words = [];
  var set, word;
  for (var i = 0; i < sets.length ; i ++) {
    set = sets[i];
    word = set[Math.floor(Math.random() * set.length)];
    words.push(word);
  }
  return words.join(' ');
}
```

ENDING A CONNECTION

You can end your half-connection at any time by calling:

```
client.end();
```

This terminates your part of the connection, but you still may get data events from the other side.

CLOSING A SERVER

If you want to shut down a TCP server, you can do:

```
server.close();
```

This stops the server from accepting any new connection, but it keeps the existing ones.

Once all the peers get disconnected, the server emits a `close` event:

05_server_close.js setup server

```
var server = require('net').createServer(function(socket) {
  socket.pipe(socket);
});
server.listen(8000, function() {
  console.log('server is listening');
});

server.once('close', function() {
  console.log('server is closed');
  clearInterval(interval);
});

/// Track connections
```

05_server_close.js track connections

```
/// Track connections
var count = 10;
var interval = setInterval(function() {
  console.log(count);
  if (count === 0) {
    console.log('shutting down...');
    server.close();
  }
  server.getConnections(function(err, connections) {
    if (connections) {
      console.log('server connections:', connections);
    }
  });
  count --;
}, 1000);
```

How can this be refactored to close when all of the connections have ended?

IDLE SOCKETS

You can terminate the connection if a socket has been inactive (no data) for a certain period:

```
var timeout = 30000; // a half minute

socket.setTimeout(timeout);

socket.on('timeout', function() {
  socket.write('idle timeout, disconnecting, bye!');
  socket.end();
});
```

The socket emits a `timeout` event, which you can then use to terminate the connection.

KEEP-ALIVE

A socket can implement a keep-alive mechanism that prevents network timeouts.

Implemented by sending an empty TCP packet requiring an ACK.

```
var timeout = 10000; // 10 seconds

socket.setKeepAlive(true, timeout);
```

This is not related to `socket.setTimeout`, which is only a local thing.

DELAY OR NO DELAY

To prevent small packets, the kernel buffers data and may wait some more time before sending it off.

(Nagle's algorithm)

This happens by default and may introduce some unwanted latency.

To turn this off, use:

```
socket.setNoDelay(true);
```

Can be reverted by:

```
socket.setNoDelay(false);
```

SUMMARY

- The net module provides you with an asynchronous TCP/IP wrapper.
- It contains methods for creating both servers and clients.
- Sockets are duplex streams