A Web of Many Requests

Requests per Page Load

My very unscientific analysis showed most sites to have 50+ requests for homepage loads. While many of these may be static content, a substantial number are dynamically generated.

Several Popular Pages

```
yahoo.com:     86 (12 HTML, 15 JS,  3 CSS)
youtube.com:   97 (16 HTML, 19 JS,  7 CSS)
linkedin.com: 146 (13 HTML, 28 JS, 13 CSS)
ebay.com:     200 ( 8 HTML,  5 JS,  3 CSS)
amazon.com:   268 (26 HTML, 48 JS, 29 CSS)
```

Common Uses for Node

- Template generation
- REST API calls
- Static content concatenation
- Metrics reporting

Common Problems

- Application becomes CPU bound
- Delivering completely static content
- External API calls are the bottleneck
- Simultaneous requests are allowed until process falters

Techniques to Help

- Don’t immediately write small packets
- Reduce the number of requests
- Use TCP for well-formatted requests
- Remember Node doesn’t gzip by default

Don’t Immediately Write

Some template modules default to “streaming mode”, calling `res.write()` for every portion of rendered template.

Instead aggregate the result, convert to a buffer, properly set the `Content-Length` header and do a single `res.end()` call.

v0.12 Note

Currently every call to `Socket#write()` makes a syscall. Upcoming is the ability to `cork()` a request, writing many buffers and sending them out as a single write.

This is enabled by default for the `http` module.

Well-formatted Requests

The format of internal API calls should be known. This allows bypassing the full HTTP parser and manually extracting the needed data.

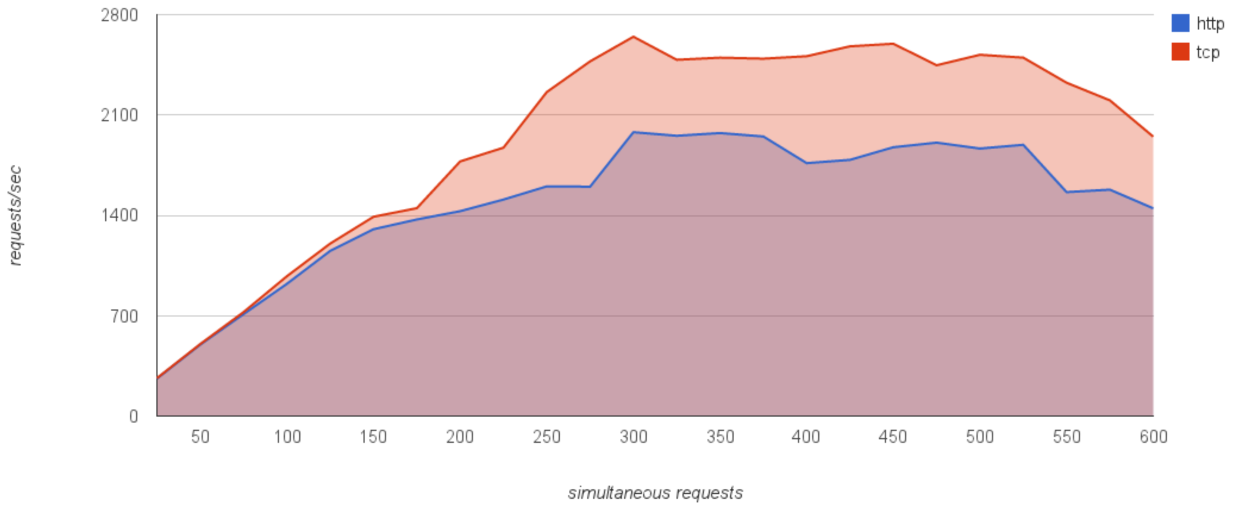
It also allows quicker detection of malformed requests so further parsing doesn’t not need to be performed.

Example TCP Replacement

```javascript
var s = net.createServer(onConnection).listen(PORT);

function onConnection(c) {
  // Disable Nagle algorithm.
  c.setNoDelay();
  // Keep open if expecting subsequent reqs.
  c.setKeepAlive(true);

  c.on('data', onData);
  c.on('error', onError);
  c.on('end', onEnd);
}
```

```javascript
function onData(chunk) {
  // Parse raw request.
  // Remember "this" is the connection instance.
}

function onError(er) {
  // Handle common errors like ECONNRESET.
}

function onEnd() {
  // Cleanup and close the socket.
}
```

Automatic vs. Manual Parsing



Reduce Number of Requests

Create a hash map of segmented static resources needed for each portion of dynamic content and write them out as a single request.

Example Storing Static Segments

```
var static_segments = {};

fs.readdirSync('resources/dir/').forEach(function(f) {
  static_segments[f] = fs.readFileSync(f);
});

// From TCP socket.
function onData(chunk) {
  // Wait on writing data until complete.
  this.cork();
  // Parsing gives us an array of resources.
  for (var i = 0; i < needed_resources.length; i++)
    this.write(static_segments[needed_resources[i]]);
  this.end();
}
```

No Default Content Compression

```
http.createServer(onRequest).listen(PORT);

function onRequest(req, res) {
  // Say "obj" is a JSON object from a REST API call.
  var msg = zlib.gzipSync(new Buffer(JSON.stringify(obj)));
  res.writeHead(200, {
    'Connection': 'keep-alive',
    'Content-Encoding': 'gzip',
    'Content-Type': 'application/json',
    'Content-Length': msg.length
  });
  res.end(msg);
}
```

gzip Performance Hit

For 250 simultaneous reqs at ~1KB response.

```
gzip'd:  980 req/sec
plain:  1470 req/sec
```

Probably more performant to use external method of content compression.

Final Thoughts

- Node has many ways to be optimized
- Don’t let “it’s fast enough” get in your way
- Be creative. Many of my best optimizations began with others saying “That can’t work.”

and now we’re done

`@trevnorris`