Guide to reducing your CDN bill using a diff tool

Same, traffic, less bandwidth, lower costs

Andrew Betts
Web Developer & Principal Developer Advocate, Fastly
A diff tool in your CDN

Same traffic, less bandwidth, lower costs

Andrew Betts
Andrew is a Web Developer and Principal Developer Advocate for Fastly, where he works with developers across the world to help make the web faster, more secure, more reliable, and easier to work with. He founded a web consultancy which was ultimately acquired by the Financial Times, led the team that created the FT’s pioneering HTML5 web app, and founded the FT’s Labs division. He is also an elected member of the W3C Technical Architecture Group, a committee of nine people who guide the development of the World Wide Web.
Requesting the difference between two previously cached files, using just a CDN configuration and a serverless cloud compute function, is a great example of exploiting edge and serverless compute services to make your website more efficient and performant, and lower your bandwidth costs.

In this article, we'll present a solution which, for a site hosting versioned downloadable assets such as software, documents, and saved games, can reduce bandwidth consumption dramatically.

Traditionally, CDNs are only useful for caching assets closer to your users. But today, modern CDNs like Fastly's can be used to perform many activities you may have previously implemented in your own infrastructure. Some of these are **products you can sign up for** as add-ons to your CDN service, while some **you can build yourself directly at the edge** by deploying your own configuration code directly to edge points of presence (POPs). With Fastly, for example, you can update your config via web interface or API to all our global edge locations in under five seconds.

**Why download data we already have?**

In the spirit of using the edge more intelligently, I was recently downloading packages using the npm package manager, and realized that although I often have a previous version of a package already installed, npm has to download the entire tarball for the new version if installing an update to a module. This seems very inefficient.

Take my open source service, Polyfill.io. It is published as an npm module, the latest version of which is 11MB gzipped, and 99MB uncompressed. Using bsdiff, we can produce a patch to summarize the changes from the penultimate version to the latest:

```
$ bsdiff polyfill_io-3.16.0.tar polyfill_io-3.17.0.tar polyfill_io-3.16.0...3.17.0.patch
$ ls -lah
 total 424
 drwxr-xr-x  5 me  staff   170B 18 Apr 15:55  .
 drwxr-xr-x 14 me  staff   476B 18 Apr 16:32  ..
-rw-r--r--  1 me  staff   209K 18 Apr 17:27 polyfill_io-3.16.0...3.17.0.patch
-rw-r--r--  1 me  staff   99M 18 Apr 15:54 polyfill_io-3.16.0.tar
-rw-r--r--  1 me  staff   97M 18 Apr 15:53 polyfill_io-3.17.0.tar
```

1. [https://www.npmjs.com/](https://www.npmjs.com/)
2. [https://polyfill.io/](https://polyfill.io/)
3. [http://www.daemonology.net/bsdiff/](http://www.daemonology.net/bsdiff/)
So if the client already has 3.16.0, getting to 3.17.0 could be done with a download of only 209KB, **a mere 1.8% of the full 11MB** (gzipped from 99MB) that you’d otherwise need for the full tarball.

However, module hosting services like npm typically store their modules on a static hosting environment like Amazon S3 or Google Cloud Storage, so there is limited or no ability to add this kind of dynamic content generation feature, and pre-generating a diff between every pair of versions of every module seems unlikely to be a good use of compute or storage resources.

Can this be done at the CDN level?

A CDN that allows origin services to be selected based on characteristics of the request could be used to route “diff” requests to a patch-generating service. With Fastly’s CDN, we can do this in VCL (Varnish Configuration Language, which we make accessible to customers). First, define a special backend:

```vcl
backend be_diff_service {
    .dynamic = true;
    .port = "443";
    .host = "<<CLOUD-FUNCTIONS-HOSTNAME>>";
    .ssl_sni_hostname = "<<CLOUD-FUNCTIONS-HOSTNAME>>";
    .ssl_cert_hostname = "<<CLOUD-FUNCTIONS-HOSTNAME>>";
    .ssl = true;
    .probe = {
        .timeout = 10s;
        .interval = 10s;
        .request = "GET /healthcheck HTTP/1.1" "Host: <<CLOUD-FUNCTIONS-HOSTNAME>>" "Connection: close"
        "User-Agent: Fastly healthcheck";
    }
}
```
Now, we can decide on a special syntax to use for patch requests, and make a small addition to `vcl_recv` that detects this syntax and routes the request to the special backend:

```vcl
sub vcl_recv {
    ....
    declare local var.diffUrlPrefix STRING;
    declare local var.diffUrlSuffix STRING;

    if (req.url ~ "^(/.*/\-\/-.*)\-(\d+\.(\d+\.(\d+))\...\(\d+\.(\d+\.(\d+))\.(tgz))\-\-\patch") { 
        set var.diffUrlPrefix = if (req.http.Fastly-SSL, "https://", "http://") req.http.Host ".global.prod.fastly.net" re.group.1 "-";
        set var.diffUrlSuffix = re.group.4;
        set req.backend = be_diff_service;
        set req.http.Host = "<<CLOUD-FUNCTIONS-HOSTNAME>>";
        set req.http.Backend-Name = "diff";
        set req.url = "/compareURLs?from=" var.diffUrlPrefix re.group.2 var.diffUrlSuffix "&to=" var.diffUrlPrefix re.group.3 var.diffUrlSuffix;
    } 
    ....
}
```

npm's downloads use URLs such as `/module-name/-/module-name-1.2.3.tgz`, so I'd like to also support `/module-name/-/module-name-1.2.3...1.2.4.tgz.patch` as a diff request. The regular expression in the VCL above captures the requests that fall into this category, and then:

1. Changes the backend to point to the diff service
2. Updates the 'Host' header so we are sending the origin's domain in the request to the service
3. Rewrites the path to match the syntax of the diff generator service

For more information on getting started with running your own VCL on the Fastly edge cloud platform, see our introductory guide to VCL.4

This is all very well, but the CDN cache nodes cannot generate diffs by themselves. This is a great use case for serverless compute services, such as AWS Lambda or Google Cloud Functions. We'll use a Google Cloud Function to handle this.
If you want to use GCF and don’t have it set up already, Google have an excellent quick start guide\(^5\) that will get you up and running. The source of the cloud function that we need looks like this:

```javascript
const url = require('url');
const zlib = require('zlib');

const fetch = require('node-fetch');
const bsdiff = require('node-bsdiff').diff;

exports.compareURLs = function compareURLs (req, res) {
  Promise.resolve()
    .then(() => {
      return Promise.all(['from', 'to'].map(param => {
        return fetch(req.query[param])
          .then(resp => {
            const name = url.parse(req.query[param]).pathname.replace(/^[^/]+/[^/]*$/,'$1');
            const isCompressed = Boolean(resp.headers.get('Content-Encoding') === 'gzip' ||
              name.match(/^(tgz|gz|gzip)$)/));
            const respStream = isCompressed ? resp.body.pipe(zlib.createGunzip()) : resp.body;
            const bufs = [];
            respStream.on('data', data => bufs.push(data));
            return new Promise(resolve => {
              respStream.on('finish', () => {
                resolve(Buffer.concat(bufs));
              });
            });
          });
      });
    });

  // Create patch and serve it
  .then(([from, to]) => {
    const patch = bsdiff(from, to);
    res.status(200);
    res.send(patch);
  });
};
```

\(^5\) https://cloud.google.com/functions/docs/quickstart
I’m using two public npm modules, node-fetch⁶ which implements the now-standard WHATWG Fetch API in NodeJS (which at time of writing is not natively supported by Node), and node-bsdiff,⁷ which performs the amazing binary diff algorithm⁸ invented by Colin Percival.⁹

This code includes no error handling or validation, and we can also improve the patch response by adding appropriate Cache-Control information (the patch can be cached for as long as the least-cacheable of the two files being compared), and also by passing through any surrogate-key¹⁰ headers present on the input files. I’ve uploaded a more comprehensive solution to GitHub¹¹ with comments, so feel free to make use of that.

Testing

To test the new endpoint, I invented differentnpm.com: a fictitious new domain name for the npm registry for which I could create a Fastly service, and I set it up with the real npm registry as its origin server. A request to download the full tarball of lodash 4.17.4, one of the most popular modules on npm, shows that the new service behaves like the npm registry:

```bash
$ curl "http://differentnpm.com.global.prod.fastly.net/lodash/-/lodash-4.17.4.tgz" -vs 1>/dev/null
< HTTP/1.1 200 OK
< Cache-Control: max-age=21600
< Content-Type: application/octet-stream
< Content-Length: 310669
< X-Served-By: cache-sjc3743-SJC, cache-sjc3628-SJC
< X-Cache: HIT, HIT
```

This request is routed to npm’s real registry, and results in a 310KB file (see the Content-Length header), and as we’d expect, is a cache HIT because this is a popular file so it’s likely to be available at the local CDN cache node.

However, this new registry also transparently supports the new diff URLs:

```bash
$ curl "http://differentnpm.com.global.prod.fastly.net/lodash/-/lodash-4.17.3...4.17.4.tgz.patch" -vs 1>/dev/null
< HTTP/1.1 200 OK
< Cache-Control: max-age=21600
< content-type: application/octet-stream
< Content-Length: 1207
< Connection: keep-alive
< X-Served-By: cache-sjc3132-SJC
< X-Cache: HIT
```

Here the request for the difference between lodash 4.17.3 and 4.17.4 is a patch of only 1,207 bytes, just 0.3% of the original size.

---

⁶https://www.npmjs.com/package/node-fetch
⁷https://www.npmjs.com/package/node-bsdiff
⁸http://www.daemonology.net/bsdiff/
⁹https://twitter.com/cperciva
¹⁰https://docs.fastly.com/guides/purging/getting-started-with-surrogate-keys
¹¹https://github.com/fastly/diff-service
Bsdiff ships with a companion bspatch tool, which can take the old file and the patch, and produce the new one:

```
$ ls -la
-rw-r--r--  1 me  staff  2254848 18 Apr 16:30 lodash-4.17.3.tar
-rw-r--r--  1 me  staff   1207 19 Apr 17:35 lodash-4.17.3...4.17.4.tgz.patch

$ bsdiff lodash-4.17.3.tar lodash-4.17.4.tar lodash-4.17.3...4.17.4.tgz.patch

$ tar tf lodash-4.17.4.tar
package/package.json
package/README.md
package/LICENSE
package/_baseToString.js
...
```

**Savings**

To work out how useful this kind of thing could be, I made a list of npm’s [most depended-upon modules](https://www.npmjs.com/browse/depended) and for each one, gathered the following data:

- Number of downloads in last month
- Size of most recent version tarball
- Size of diff between most recent and penultimate version tarball

One thing we can’t know from public data is how often a user has a prior version of a file in cache locally. Let’s look at the impact if that number were 5%, 15%, and 50%:

<table>
<thead>
<tr>
<th>Module</th>
<th>Downloads (1000s)</th>
<th>Size (bytes)</th>
<th>Monthly transfer (GB)</th>
<th>Patch size</th>
<th>Monthly data savings, GB, by cache ratio:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Abs (b)</td>
<td>Rel (%) 5% 15% 50%</td>
</tr>
<tr>
<td>lodash</td>
<td>42,866</td>
<td>310,669</td>
<td>12,403</td>
<td>1,207</td>
<td>0.39% 618 1,853 6,177</td>
</tr>
<tr>
<td>request</td>
<td>24,756</td>
<td>56,636</td>
<td>1,306</td>
<td>3,248</td>
<td>5.73% 62 185 615</td>
</tr>
<tr>
<td>async</td>
<td>43,923</td>
<td>97,968</td>
<td>4,008</td>
<td>23,083</td>
<td>23.56% 153 459 1,532</td>
</tr>
<tr>
<td>express</td>
<td>11,577</td>
<td>52,372</td>
<td>565</td>
<td>602</td>
<td>1.15% 28 84 279</td>
</tr>
<tr>
<td>chalk</td>
<td>21,045</td>
<td>5,236</td>
<td>103</td>
<td>1,027</td>
<td>19.61% 4 12 41</td>
</tr>
<tr>
<td>bluebird</td>
<td>14,327</td>
<td>135,089</td>
<td>1,803</td>
<td>2,669</td>
<td>1.98% 88 265 883</td>
</tr>
<tr>
<td>underscore</td>
<td>12,229</td>
<td>34,172</td>
<td>369</td>
<td>6,879</td>
<td>20.13% 16 47 155</td>
</tr>
<tr>
<td>commander</td>
<td>26,118</td>
<td>13,425</td>
<td>327</td>
<td>1,309</td>
<td>9.75% 15 44 147</td>
</tr>
<tr>
<td>debug</td>
<td>45,226</td>
<td>16,144</td>
<td>680</td>
<td>588</td>
<td>3.64% 33 98 328</td>
</tr>
<tr>
<td>moment</td>
<td>9,219</td>
<td>497,477</td>
<td>4,271</td>
<td>891</td>
<td>0.18% 213 640 2,132</td>
</tr>
<tr>
<td>Total (top 10 modules)</td>
<td>251,286</td>
<td>25,853</td>
<td>1,229</td>
<td>3,687</td>
<td>12,290</td>
</tr>
</tbody>
</table>

Relative saving 4.75% 14.26% 47.54%
Diff sizes obviously vary, and the most popular npm modules also tend to be quite small, but if 50% of
npm’s module requests could be diffs, then this data suggests that they would eliminate almost that same
percentage of their bandwidth.

Conclusion

Package managers are not the only type of business that could benefit from this. Android uses binary diffs
to update apps from the Google Play store, and any scenario where you need to send a user an update to
something they already have, diffs can make your bandwidth use dramatically more efficient.

Whether or not your business can make use of this kind of solution, there are many ways you can get more
value from your CDN. Whatever your scaling challenge, a globally distributed edge compute and caching
network is often a key part of the solution, so choose one that gives you as much control and flexibility as
possible.