HTTP/2 Compression

(or at least one proposal)

Problems with currently deployed (gzip) compression

- Moderate CPU expense for the sender
- Double this cost for a proxy
- CRIME exploit

Delta-encoding plus canonical static huffman

That is what is being proposed here.

So.. how it works...

Detailed "How It Works"

The basic storage mechanism for this compressor is that of an LRU of Key-Value pairs.

The compressor includes a set of operations which exist to manipulate this LRU by either inserting a new key-value, or by creating a new key-value by reusing an old key with a new value.

Detailed "How It Works" (cont'd)

Additionally, the compressor includes a set of pointers to these key-value lines in the LRU.

These pointers are maintained and modified by another operation.

Each such set of pointers to key-value pairs represents a set of headers as would appear in a request or response.

Detailed "How It Works" (cont'd)

The compressor is pre-seeded with the common keys and some common values

It is very, very rare that a key is referenced by other than a numeric ID.

How well does this work?

At a moderate gzip window size (as deployed in Chrome for SPDY): both gzip and this achieve the same compression ratio over my dataset.

With a large gzip window size, gzip achieves a 90% deflation, whereas the proposed compressor achieves 85% deflation with 1/3rd the CPU cost.

How well does this work? (cont'd)

As far as we can tell (so far), this compression scheme is not vulnerable to the CRIME exploit.

Disadvantages with this compressor

It doesn't achieve the maximum possible compression ratio*

*probably-- it is possible that, if clients and servers change how the send headers, the delta-compressor would always do better than gzip or equivalent but this is not proven.

Advantages of the compressor

It is fast. 3X cheaper than gzip and it could probably be faster if better optimized

It is proxy-friendly: Most of the decompressor state can be reused (with the help of an ID mapping) on the compression side. Thus, a proxy which wishes to maintain compression on both ingress and egress sides can do so cheaply

Advantages of the compressor (cont'd)

It is proxy-friendly (cont'd): Proxies also have available a don't-compress mechanism, whereby they can send interpretable data which doesn't affect, nor refers to compressor state.

Advantages of the compressor (cont'd)

Since we're encoding state changes, we don't have to 'reconsitute' the headers each time--we can instead simply change the server state as appropriate

Since huffman-encoded strings are encoded using a static encoder, string matching can be done on the encoded form. No decoding is necessary before interpretation.

Advantages of the compressor (cont'd)

Most of the time, keys are referred to by ID and rarely take up space on the wire. This compares favorably (and is analogous) with the Friendly I-D, but doesn't require maintenance of a registry.

Avenues of further research:

Since the huffman-coding is a canonical huffman code, it is theoretically possible to receive a new huffman encoding efficiently Should it be done?

It is also theoretically possible to send a new 'initial dictionary' to be used in future versions. Should it be done?

Avenues of further research:

Should the ID space encode distance in characters consumed instead of ID of key-value pair?