Clarifying the Fundamentals of HTTP

Jeffrey Mogul
Compaq? Western Research Lab
JeffMogul@acm.org

May 1, 2002

2002 International World Wide Web Conference

What's wrong?

Myth: the HTTP protocol is

- Simple
- Easy to understand
- Easy to extend

Reality: the HTTP protocol is

- Complicated (RFC2616: 176 pages, + other RFCs)
- Confusing
  - many mailing-list questions by implementors
- Hard to extend without conflicts

Goal of this talk

Examine fundamental definitions and models of HTTP

- Look at current models/definitions
- Show where these cause trouble
- Fix them

Not “write model first, then make HTTP fit the model”

Rather:

- deduce sound model, based on HTTP experience
- then, make sure protocol fits its implied model

Outline

- Protocol or distributed system?
- HTTP's data type model
- HTTP's data access model
- Extensibility
- Other stuff
- Related and future work

Is it a protocol or a distributed system?

Protocol designers:

- Think in terms of messages
- Treat implementations as black boxes
- Worry about interoperability & extensibility

Distributed systems designers:

- Think in terms of state
- Think hard about caches
- Worry about correctness and error conditions

We need to think *both* ways about HTTP

The importance of caching

HTTP: as a network protocol and a distributed system

- Depends on caching for performance

Caching illuminates weaknesses in any specification:

- Caching must be semantically transparent
  - Wrong answers are worse than slow answers
  - Bad caches encourage cache-busting
- Caches are intermediaries, not end-points
  - Forces us to be explicit about the details
  - Can become inhibitors of extensibility

Outline

- Protocol or distributed system?
- HTTP's data type model
- HTTP's data access model
- Extensibility
- Other stuff
- Related and future work

Data type model

What is a “data type model” for HTTP?

- Data types operated upon by the protocol
- Also includes transformations between data types
- Not: MIME types (these are opaque to HTTP)

Convenient to think of “message generation pipeline”

- Abstraction of steps carried out by server
- Nodes represent data types
- Edges represent transformations

HTTP's existing data type model

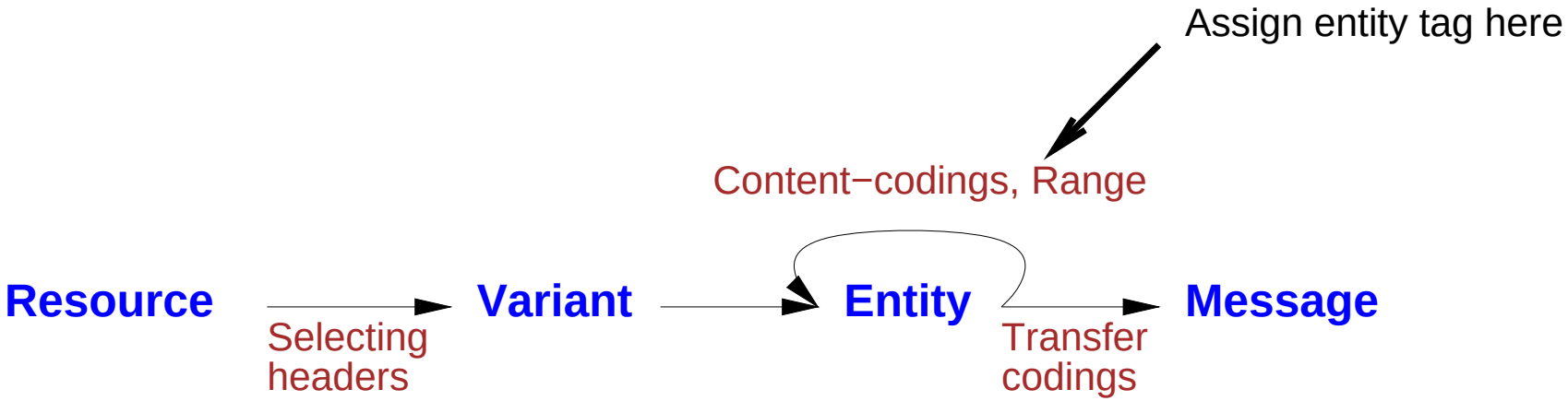
Fuzzy model, implicit in the spec:

- **Resource**: thing which a URL points to
- **Variant**: language-specific version of resource
- **Entity**: information (body + headers) in a response
- **Message**: carries entity or pieces thereof

Also, **Entity tag**:

- Assigned & sent by server, stored in cache entry
- Sent by client in conditional request (revalidation)
- If matches, cache entry still valid

Message generation pipeline



Things to note:

- There's a cycle in the pipeline
- Entity tag is assigned on an edge, not at a node
- Edge between variant and entity is not labeled

Problems with existing model

Specification of caching

- What does a cache store?
  - Not any of: resource, variant, entity, message

Partial results

- How to combine ranges, compression?
- How to define delta encoding?
- When does the server assign an entity tag?

These are both too complex to describe here!

More problems with existing model

Header categorizations

- What headers can a proxy cache modify?
- How to make this extensible?

Protocol specification complexity

- Lack of modularity
- Complex wording

An improved data type model

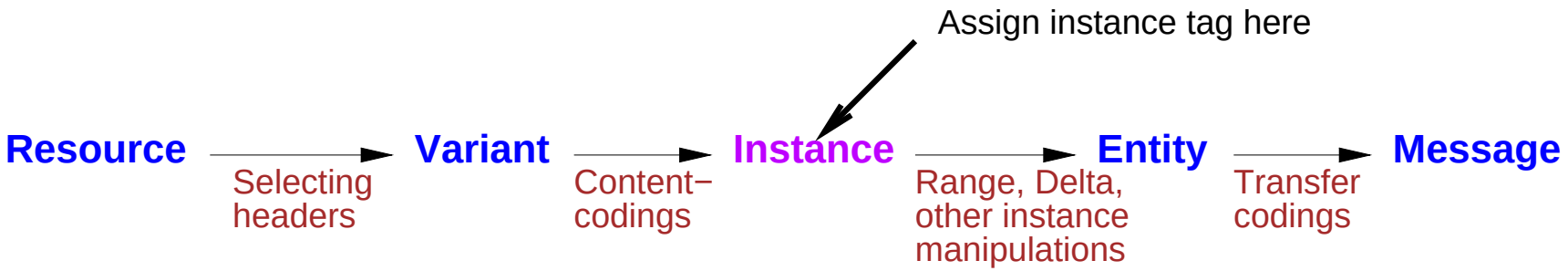
Key idea: add one new data type:

Instance: the entire result of successfully applying GET to a given resource variant at a given point in time. *(Defined more formally in the paper)*

Also add (optional) instance manipulations

- May generate partial results
- E.g., Range selection, Delta encoding, compression
- Add new `IM` and `A-IM` headers for labeling

Improved message generation pipeline



Things to note:

- No cycles in pipeline
- “Entity tags” are really instance tags!
- Entity/instance tag is assigned at a node
- All edges are labeled

Why is the new model better?

Caching specification is much simpler

- Cache entries store instances (or parts thereof)
- “Entity tags” identify instances

Much cleaner handling of partial results

- Clients can specify ordering (using `A-IM`)
- Composition with compression is well-defined

Header categorizations make sense (mostly)

- Many more categories than RFC2616 (see paper)
- Rules for, e.g., “end-to-end” hdrs (almost) trivial

Outline

- Protocol or distributed system?
- HTTP's data type model
- HTTP's data access model
- Extensibility
- Other stuff
- Related and future work

Access model

What is an “access model” for HTTP?

- *What* kinds of data can be accessed
  - E.g., mutable or not?; side-effects?; idempotent?
- *How* data is accessed
  - E.g., GET, PUT, POST, DELETE
- Not an “access-control” (protection) model

Most important for

- Non-human agents (robots, automated clients)
- Intermediaries (caches, proxies)

Problems with the access model

Too much is *inferred* that should be *explicit*:

- No access-model labeling, e.g.:
  - is it safe to replay the request?
  - will there be a new instance value in the future?
- No clear mapping to protocol requirements, e.g.:
  - can I do a PUT on this resource?
  - is it safe to pipeline requests on this resource?
- “Static” vs. “dynamic” inferred from `?': nonsense
  - False belief: implies “no-cache” and “no-replay”

Fixing the access model

Add labels

- Make everything explicit
- Never require inferences

Labels include

- “mutable”, “side-effects”, “assignable”, etc.
- Cost(s) to generate response (rather than “dynamic”)

Attach labels to instances, resources (maybe variants?)

Outline

- Protocol or distributed system?
- HTTP's data type model
- HTTP's data access model
- Extensibility
- Other stuff
- Related and future work

Extensibility

HTTP has one simple, powerful extension mechanism:

- “Ignore all headers you don't understand”

But lacks good support for complex extensions:

- Do we both agree to use a given extension?
- Do we agree what it *means*?
- Does entire path (including proxies) support it?

Extensibility problems

Consequences of current shortcomings include:

- Servers basing their responses on `User-agent`
- Javascript that “knows” what each browser can do

Failed attempts at solutions include:

- OPTIONS method
  - no useful syntax or semantics
- RFC2774 “HTTP Extension Framework”
  - too complex
  - stuck as an “experimental” RFC

Why HTTP version numbers don't help

HTTP messages include a version number, but:

- Too many optional features associated with version
- Many different revisions for each version
- Some proxies lie about the version number
- Hop-by-hop only; says nothing about full path

What are the possible solutions?

How do we discover extensions?

- Trial and error
- Negotiate
- Declare capabilities

How do we name extensions?

- Implicitly (trial and error)
- Decentralized and flexible (as in RFC2774)
- Centralized and slow to change

A proposed extension mechanism

Choose:

- Declare capabilities (of every hop)
- Centralized naming

Use RFC numbers as extension names:

- Compact, non-ambiguous namespace
- RFCs are immutable (by IETF rules)
- IETF requires unambiguous meaning for RFCs
- Define “profile” RFCs for compact naming of sets
- (Proposed 1997 by J. Cohen, S. Lawrence, & me)

Outline

- Protocol or distributed system?
- HTTP's data type model
- HTTP's data access model
- Extensibility
- Other stuff
  - Variants – see the paper
  - Intermediaries – see the paper
  - Protocol support for user-interface concerns
- Related and future work

Protocol support for user-interface concerns

Theory:

- clear boundary between HTTP protocol and UI
- (after all, some HTTP clients have no UI!)

Reality:

- server & client need to communicate about UI state
- already some spec. words re: security state

Improving UI support

Example: history mechanisms (“back” & “forward”)

- Spec. distinguishes these from caching
- But most implementations blur the distinction
- Forces annoying site-design glitches
  - E.g., “do not use the `back' button” warnings
  - E.g., `no-store` for “security” on shared browsers

Better approach: explicit server control over UI

- Separate from cache-related features
- E.g., “prevent this page from appearing in history”

Outline

- Protocol or distributed system?
- HTTP's data type model
- HTTP's data access model
- Extensibility
- Other stuff
- Related and future work

Related work

Fielding & Taylor (2000, Intl. Conf. on Software Eng.)

- *"Principled design of the modern Web architecture"*
- Idealized model for interactions on the Web
- Points out some flaws in HTTP
- "Representation" model blurs too many distinctions

Eastlake (2001, Internet-Draft)

- *"Protocol versus document points of view"*
- Argues against taking just one of these points of view

More related work

Baker (2001, Internet-Draft)

- *"An abstract model for HTTP resource state"*
- Effectively a data access model for HTTP/1.1
- Improves clarity without actually changing spec.

Several "HTTP Next Generation" efforts:

- Usually attempts to redesign HTTP from ground up
- None have gone very far

Future work

Data type model needs:

- Testing on other extensions
  - e.g., CDNs, coherent caching
- More work on arcane header-specific rules

Unfinished business:

- Variants (naming, semantics, and caching)
- Extension model (naming and semantics)
- Transcoding (and intermediaries in general)

Clean up the spec, in general!

Messages to take home

The HTTP spec needs some cleanup:

- Leave the *protocol* alone, but fix the *words*
- Explicit, careful models help with rigor and clarity
- “Entity tags” are really “instance tags”
- Think harder about how composition of features

Eliminate ambiguity:

- Never require inferences or heuristics
- Add tagging where it's necessary
- Extensions need a simple, precise namespace