

Architecture and pragmatics of server-directed transcoding

Björn Knutsson

DSL, University of Pennsylvania

In cooperation with Honghui Lu and Jeffrey Mogul

Introduction

- Web client diversity is increasing
 - Bandwidth and other network properties
 - Wireless, modem, Cable/ADSL
 - Display capabilities (size, colors)
 - Internet enabled mobile phones, PDAs, WebTV
- Increasing need to **adapt** content to
 - Fit display characteristics
 - Reduce latency and bandwidth cost
- Don't want to diminish experience for “power” users

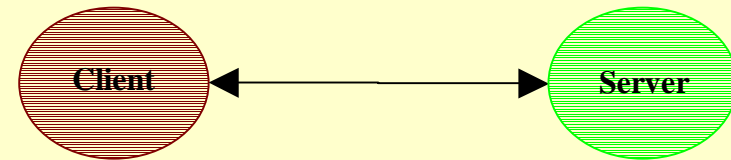
How do we adapt content?

- Idea is to adapt content by **transcoding**
 - Take existing content and apply various algorithms to transform content into a more appropriate form
- Simple example: Resizing image



Where do we transcode?

- At client:
 - Must still transfer original image – cannot deal with bandwidth cost or latency
 - Requires CPU resources at client
 - Risk of semantic losses



- At server:
 - Static approach does not scale well
 - Dynamic approach increases CPU load on server
 - Both cause problems for web caching

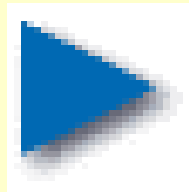
Neither model is satisfactory!

Dealing with semantic losses

- Semantic losses happen because the transcoder does not know the semantic of the content
- “Blind transcoding” - result may be worthless
- Need server hints to limit semantic loss

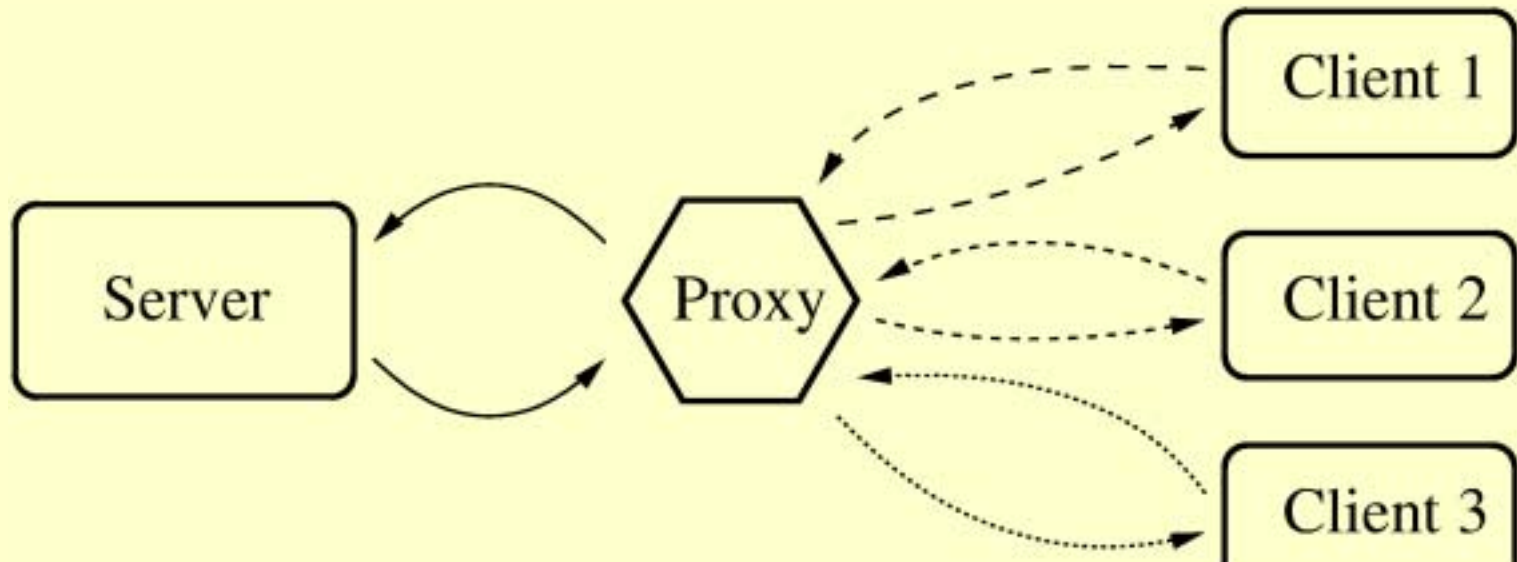


Reduce byte size



Effects on web caching

- Server content adaptation cause caching problems
 - Static versions means each version must be cached
 - Dynamically generated are hard to cache
 - Can cause cache to return wrong version
- Need to co-locate transcoding with caching



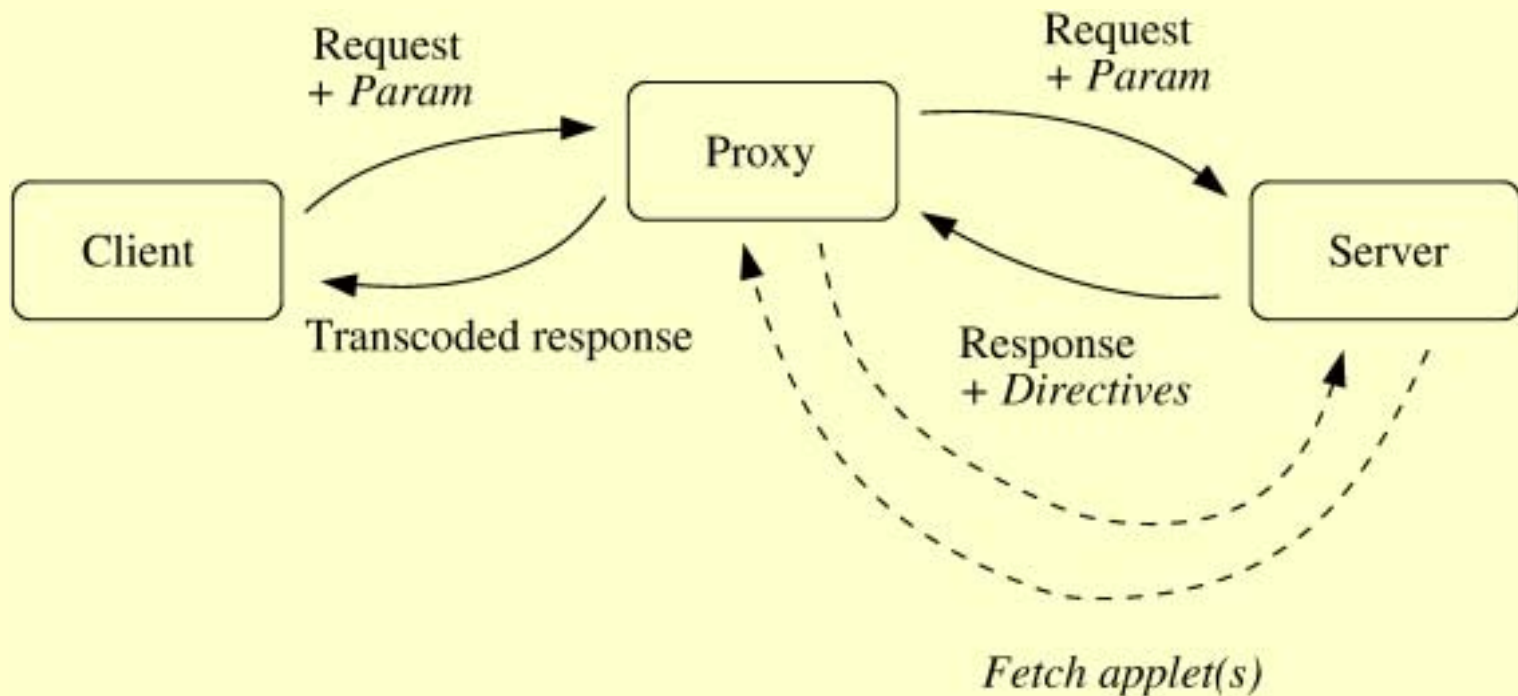
Our solution: Hybrid proxy

- Traditional proxy deals with the CPU/bandwidth problem of client solution, not the semantic issue
- Our hybrid proxy takes Server Directives from the content provider that describe how the object can be safely transcoded without semantic losses
 - Same principle as dynamic server transcoding, but decoupled from the server, allowing more flexibility



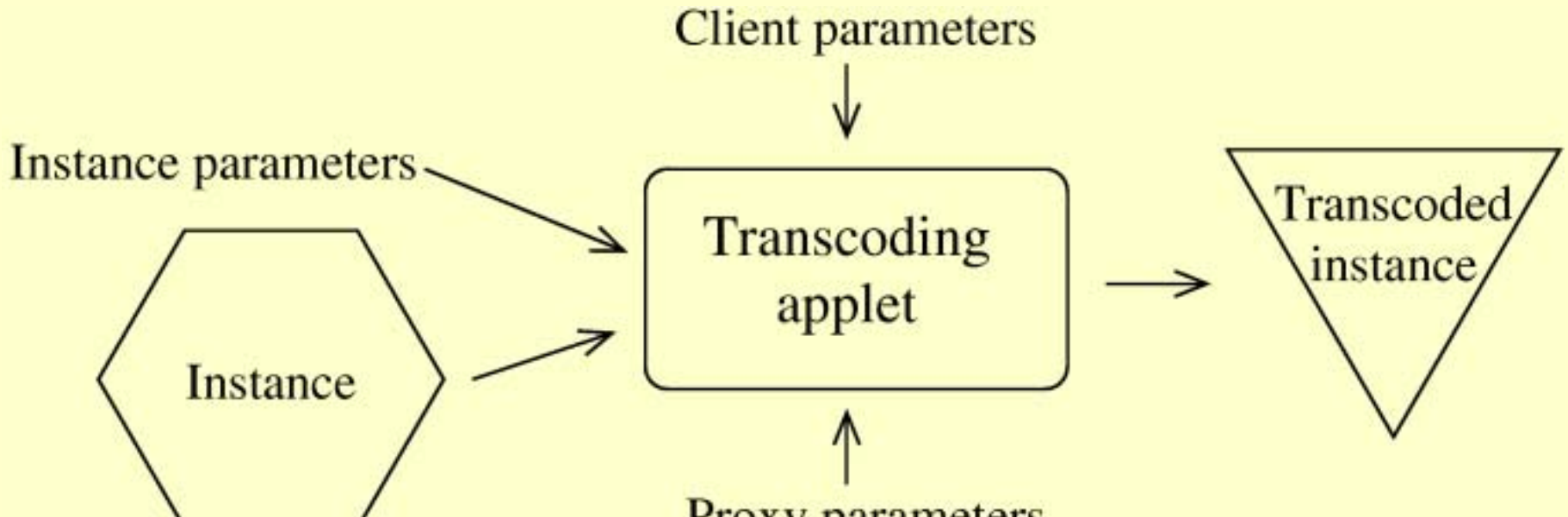
Architecture

- Client parameters and server directives together decide what transcoding to apply
- Our model – Server Directives are code (applets)
 - Provides additional flexibility and further decoupling



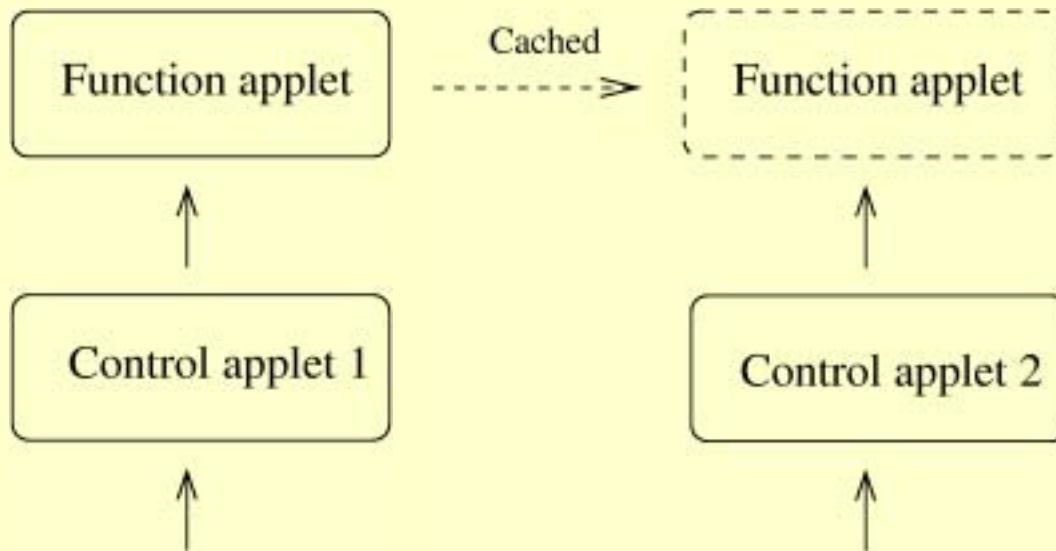
Applet environment

- Transcoding applet takes parameters from:
 - Server – per-instance, simplifies applet re-use
 - Client – describes target for transcoding
 - Proxy – describes resource constraints



Applet re-use

- We expect most applets to be generic
 - Content designers are not necessarily programmers
 - Cost of writing transcoding applets
 - Ease of constructing generic applets
- Simple content specific applets *can* be built from reusable components



Example applet: 'crop-interp'

- Designed for images with one important subject and surroundings that just add flavor
- Use set of crop boxes and interpolate between them to fit arbitrary sizes
- Re-usable applet, you only need to specify a set of crop boxes to use for new image



crop-interp' vs blind transcoder

- Crop is a CPU efficient transcoding operation
- Cropping is not possible with blind transcoder
- Quality is hard to quantify – subjective measure
- Example – original is 66KB, 600x800 image:



5830 bytes 170ms



5815 bytes 276ms



5831 bytes 86ms

Whole system benchmarks

- Prototype system unoptimized – no caching
- Applets are small (~2KB) – download quickly
- Time to run control applet was ~8ms, bulk of proxy runtime is transcoding operations
 - Need native implementations for efficiency
- Significantly decrease latency for low-bandwidth networks (1.3x – 8.6x)
- At xDSL speeds (768Kbps), results range from 0.8x to 2.4x, depending on target size
 - Transcode to fit device, not to reduce latency

Summary

- Server Directed Transcoding allow us to combine the advantages of server and client/proxy transcoding
- Our implementation of SDT uses “mobile” code and integrates with web cache
- SDT allow us both to do a better job as well as offering opportunities to do it at a lower cost
- Architecture is generic – can transcode all types of content, and can also fall-back to “blind transcoding” when necessary
 - Allows for incremental deployment of SDT

Future work

- A complete implementation is underway
 - Java as control language (Perl in prototype)
 - Integration with existing web cache (squid)
- Looking at proxy resource management
 - How to deal with proxy overload
 - Incentives for client quality compromises
 - Market mechanisms for buying superior service
- Application of results on other domains
 - Decoupling of client/server is good