HTTP & SOCKS

Christian Grothoff
`christian@grothoff.org`
`http://grothoff.org/christian/`

"We shape our buildings; thereafter they shape us." –Winston Churchill

HTTP

- Text-based protocol (ASCII, LFLR)
- HTTP 1.0 and HTTP 1.1 as main variants
- Client: “METHOD URL HTTP/1.X”, headers, body, footers
- Server: “STATUSCODE STATUSMESSAGE”, headers, body, footers

HTTP Methods

- GET — request data, idempotent
- HEAD — request data, header only, idempotent
- POST — send request, once-only
- PUT — send request, idempotent
- DELETE — delete data, idempotent
- CONNECT — for proxies
- OPTIONS — query capabilities
- TRACE — debugging

Header & Footer format

- Key-value pairs
- Format: “KEY: VALUE” + LRLF
- Value can continue on next line (leading space)
- ASCII encoded

Headers, Body & Footers

- Headers terminated by empty line (LFLR)
- Headers can specify body length: “Content-Length”
- Headers can specify “chunked encoding”
- If neither is specified, body ends at EOF
- Footers follows body

Chunked encoding

- Body consists of “chunks”
- Each chunk begins with HEX-encoded length prefix
- HEX encoding at most 6 bytes (ABCDEF)
- Followed by LRLF and then the body data
- End-of-file marked with length-prefix of 0

Pipelining

- Allows client to send 2nd request over same TCP connection
- Most clients only do this server completed response, if at all
- “Connection: close” disables pipelining
- HTTP 1.1-only

Cache Control

- Age, Cache-Control, Date, ETag, Expires, Last-Modified, Retry-After
- If-Match, If-None-Match, If-Range, If-Unmodified-Since

Client Capabilities

- Accept-Charset
- Accept-Encoding
- Accept-Language
- Accept-Ranges
- Use-Agent

Request Details

- Cookie
- From
- Host
- Referer
- Via

Response Details

- Content-Encoding
- Content-Language
- Content-Location
- Content-MD5
- Content-Type

Response Status Codes

- 1xx — Informational
- 2xx — Successful
- 3xx — Redirection
- 4xx — Client Error
- 5xx — Server Error

Summary

HTTP features:

- Pipelining (but often not used by browsers)
- Caching
- Compression of body (optional)
- Authentication (basic, digest, HTTPS)
- Encryption (with HTTPS)
- Integrity protection (with footers)

HTTP 2.0

- Developed by Hypertext Transfer Protocol Bis (httpbis) working group
- Connection multiplexing, header compression, request-response pipelining
- Same transaction semantics as HTTP 1.1
- Drafts: Google SPDY, Microsoft HTTP Speed+Mobility, Tarreau-FOSS draft
- Main goal: latency, latency, latency

Key Ideas of SPDY/HTTP 2.0

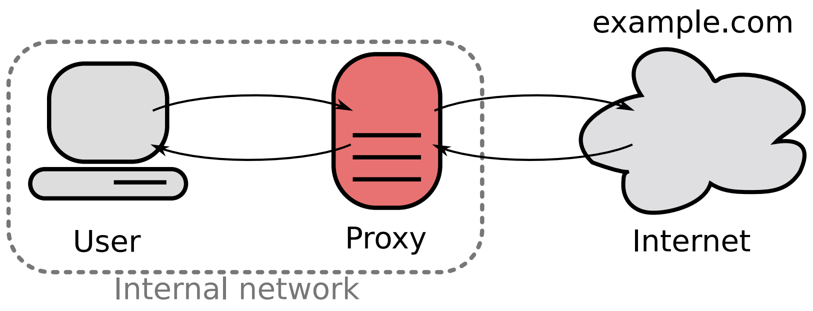
- Avoid redundant, text-based headers, go binary, compress, transmit once
- Multiplex multiple requests and responses concurrently over the same TCP stream
- Allow client to prioritize requests
- Allow server to push responses for which there is not request yet

Proxies

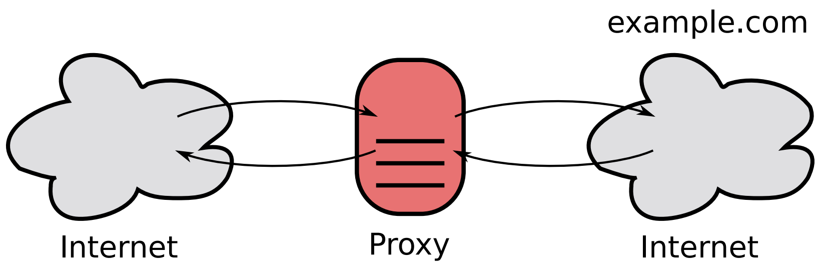
Proxies are servers that act as an intermediary. Uses:

- Security: anonymity, access control, access control circumvention, logging/auditing
- Performance: caching, multiplexing

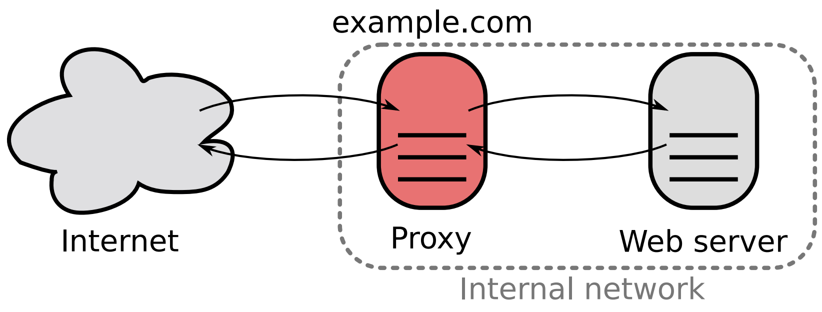
Forward Proxy



Open Proxy



Reverse Proxy



SOCKet Secure (SOCKS)

- Used for Forward or Open Proxies
- Supported by virtually all HTTP-client implementations
- Very simple protocol

$\Rightarrow$ Suggested type of proxy for the project!

SOCKS4 request

- Version — 0x04
- Command code — 0x01 = stream, 0x02 =port
- Destination port — 2 bytes, NBO
- Destination IP — 4 bytes, NBO
- User name — 0-terminated string
- Followed by data to send to the server

SOCKS4 response

- Padding (1) — 0x00
- Status — 0x5a = granted, 0x5b = rejected
- Padding (2) — 2 bytes
- Padding (3) — 4 bytes
- Followed by data from the server

SOCKS4a

- Allows client to delegate DNS resolution to proxy
- IP address beings with 3x 0x00, followed by one byte that is not 0x00
- User name is followed by 0-terminated hostname
- Server response contains port and IP in padding 2 and 3

SOCKS5

+ Authentication
+ IPv6 support
+ UDP support
- Slightly more complicated wire format

$\Rightarrow$ Use SOCKS4(a) use for the project!

Questions

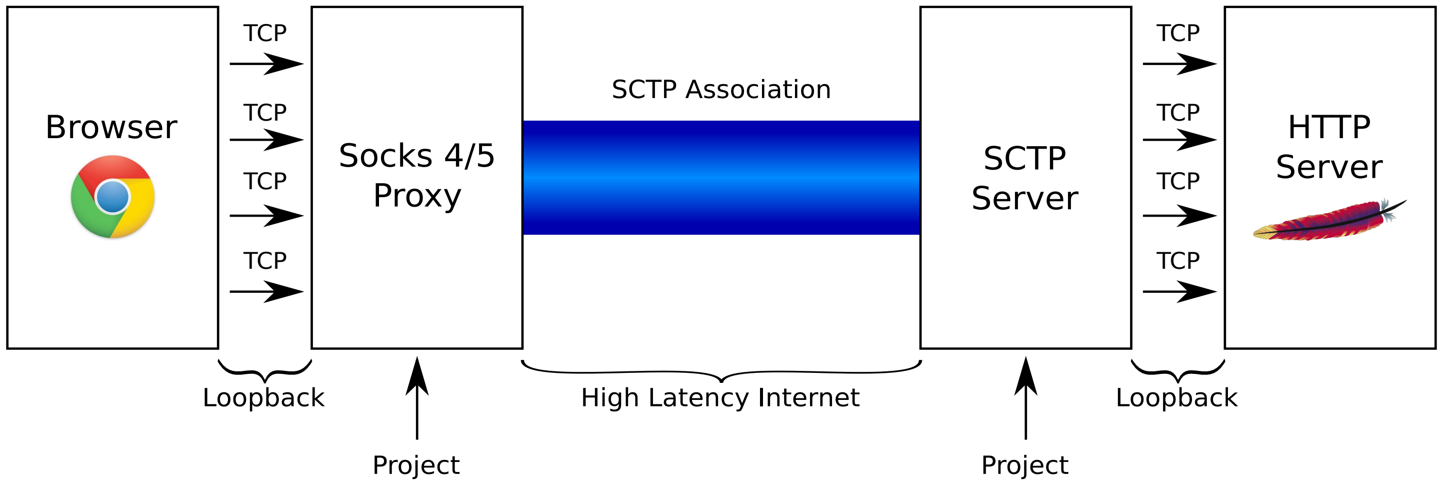
?

“” –

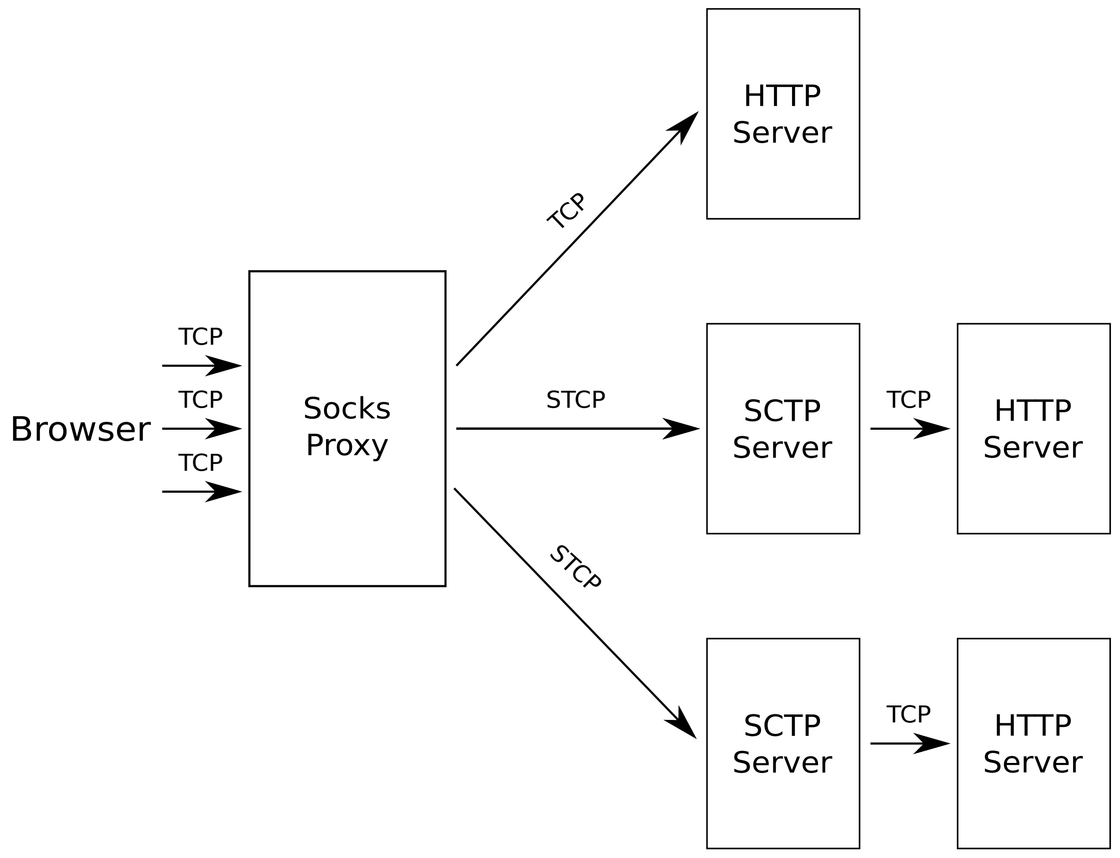
Questions

?

The Project: HTTP over SCTP



The Project: HTTP over SCTP



fsnsg

Step 1: Keep It Simple

- One association (UDP style?) from proxy to SCTP server
- One stream per HTTP request
- Grab SOCKSv4a data, rest passes through untouched
- SCTP server not running $\Rightarrow$ use TCP

Step 2: Benchark

- Setup testbed
- HTTP server with “complex” website
- Browser downloading full site with parallel TCP connections and HTTP 1.1 pipelining
- Measure: latency, bandwidth consumption, etc.
- Simulate congestion, high delay
- Visualize

Step 3: Optimize

- Idea: eliminate duplicate HTTP header transmission
- Idea: enable server push
- Idea: add compression
- Tools: GNU libmicrohttpd can provide HTTP server in proxy
- Tools: libcurl can implement HTTP client
- Tools: libtidy can parse HTML
- Tools: libz, libbz2 can compress data

Step 4: Compare

- HTTP over TCP
- HTTP over SCTP
- SPDY

Questions

?

“There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.” – C.A.R. Hoare

RTFL