HTTP offload is a ~~dumb~~ great idea whose time has come

Charles Eckman[*]
Stephen Longfield

Abstract

Since time immemorial, providers of hypertext services have dreamed of making their servers better. One effective technique for accomplishing making software faster and stronger is to implement it *harder*,[13] that is, to offload processing from a CPU into a dedicated hardware device. This technique has been used for lower layers of the network stack, but for hypertext, they have been mere fantasies, met with unjustified disdain. We show that HTTP offload is an effective technique for addressing truly vital, very specific goals, including performance, efficiency, and preparation of hot beverages.

Just like other ideas whose time has come[12], now it is HTTP's turn to be offloaded into hardware.

CCS Concepts

• **Information systems → RESTful web services**; • **Hardware → Hardware accelerators**.

Keywords

FPGA, HTTP, Good Ideas

1 Introduction

Originally, "computing" was something done in a person's head. Over time, the computing industry has steadily offloaded computation from general-purpose computers (e.g. brains) into specialized hardware artifacts (e.g. Napier's bones, bomba kryptologiczna, Furby). In recent years, specialized hardware has provided acceleration for bitcoin mining (SHA hashing), machine learning (matrix-multiply units), and video encoding (hardware codecs).

Accelerating network protocols has proven a fruitful domain for acceleration- focusing on a common horizontal, rather than a vertical. Today's high-end NICs and routers have dedicated hardware for Ethernet PHY, IP forwarding, TCP offloading, and even TLS offloading.

To our knowledge, however, network acceleration has stalled out at OSI layer 6.[1] While running HTTP on embedded devices is common [5], "offload" onto another processor does not provide the full benefits of hardware acceleration. We set out to complete the walk up the OSI stack, and create an HTTP server in hardware.[2]

2 Background

2.1 Fomu

The specific hardware we chose to use as the platform for our HTTP implementation was the Fomu[11] platform, chosen as the authors already owned them.

[*]This was Charles' idea.
[1]Not that we looked very hard.
[2]Due to time and budget constraints, "hardware" means an FPGA. Anyone want to sponsor us for Tiny Tapeout?[17]

This platform contains an iCE40UP5K FPGA, which is well supported by open source toolchains. It has RGB LEDs, which are essential for the blinking light needed on every proper peice of hardwares [3]. The serial-over-USB interface is supports isn't perfect for HTTP, but the authors make it work.

Apart from all of these conveniences, the Fomu has the added advantage of being short and stout, which will come in handy later.

2.2 Amaranth

The hardware definition was implemented using Amaranth HDL[18], selected for its code readability and virtue of appearing far earlier in alphabetical listings than competitors Verilog and VHDL.

This language contains many useful primitives, such as ready-valid channels, as well as a built-in simulator supporting unit testing of modules.

As the Amaranth language is built on top of Python, we were able to integrate with the Hypothesis[8] testing library to get property-based testing, and py_test[7] for unit testing.

3 Engineering

The Fomu device only has USB for input and output. As such, we did not implement all the network layers up to HTTP. Here we describe the portion of the transport layer used to bridge the Internet to the Fomu, and handling of the HTTP in hardware.

3.1 nTCP

In our HDL design, we began with the LUNA USB stack [6] because it was the first USB stack we could get working. We configured the Fomu to present itself as a USB serial device (USB CDC ACM FTW).

While HTTP/1 and HTTP/2 are designed to run over TCP, a TCP connection is not quite the same as a serial stream. A TCP connection includes explicit setup and teardown messages, allowing each party to detect the start and end of stream; HTTP/1.0 ([15]) makes use of this to find the start and end of each request. Lacking these brackets, HTTP/1.1 is subject to request smuggling,[2] a security failure which precludes all serious websites from using HTTP/1.1.

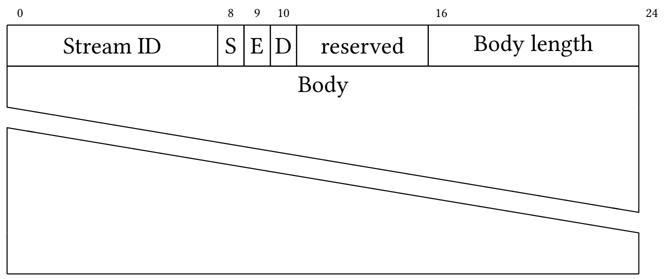


Figure 1: nTCP packet layout

We therefore decided to adopt HTTP/1.0 semantics ("one request per connection"), and built a small protocol called "Not TCP" on

HTTP Request	**HTTP Response**
`POST /led HTTP/1.0\r\n`	`HTTP/1.0 200 OK\r\n`
`Host: test\r\n`	`Host: Fomu\r\n`
`User-Agent: chartreuse.org\r\n`	`Content-Type: text/plain; charset=utf-8\r\n`
`Content-Type: text/plain\r\n`	`\r\n`
`\r\n`	`Thank you!\r\n`
`7FFF00\r\n`	

Table 1: HTTP/1.0 Request and Response Pair

top of the serial line. As shown in Figure 1, each Not TCP packet consists of a 3-byte header followed by a variable-length body. The header provides a stream ID, allowing multiplexing requests to the device; flags, indicating the start (S), end (E) and direction (D) of the request/response; and the length of the body following the header.

A program running on the USB host provides higher-level networking connectivity (TCP acceptance) and bridges the resulting request and response streams to the Fomu device over Not TCP. Put more simply, the host program is a logic inverter: it turns TCP into Not TCP and vice versa.

3.2 HTTP Parsing in Hardware

Without the persistent connections of later revisions, HTTP/1.0 can be implemented in a pure dataflow manner, with no need to maintain state between nTCP transactions. This allowed the authors to simplify their implementation to fit in the LUT constraints of the Fomu, and the time constraints of publication.

Additionally, in HTTP/1.0, headers are just strongly encouraged, with none being strictly required. We take advantage of that by completely ignoring headers in requests, and only producing them in responses when we feel like it.

Figure 2 shows the dataflow diagram, where double-lines are used to show ready-valid channels that carry individual characters.

One of our endpoints allows users to `POST` a hex color to the Fomu's LEDs. An example request/response might look like Table 1, which would set the LEDs to a brilliant shade of green.[3]

In this example, the start line parser would validate the start line, and pull out the `POST` method and `/led` path. The headers would be summarily discarded, and the remainder of the message would be forwarded to the LED body parser to pull out the colors and decide on a response. Once the response has been sent to the output channel, the nTCP session would end and the HTTP module would be ready for another request.

More complex responses (e.g., the `/count` endpoint, which responds with internal diagnostic counts) require more data to flow from the body parser to the responders. Drawing those lines in Figure 2 is left as an exercise for the reader. Crayons may be provided upon request.

3.3 RFC2324 and RFC7168

The IETF defined the Hyper Text Coffee Pot Control Protocol, HTCPCP/1.0 in RFC2324[9], and expanded on it in RFC7168[14].

[3]In practice, the Fomu's red channel is much stronger than the green channel, resulting in a sweeter, softer, lower-ABV yellow.

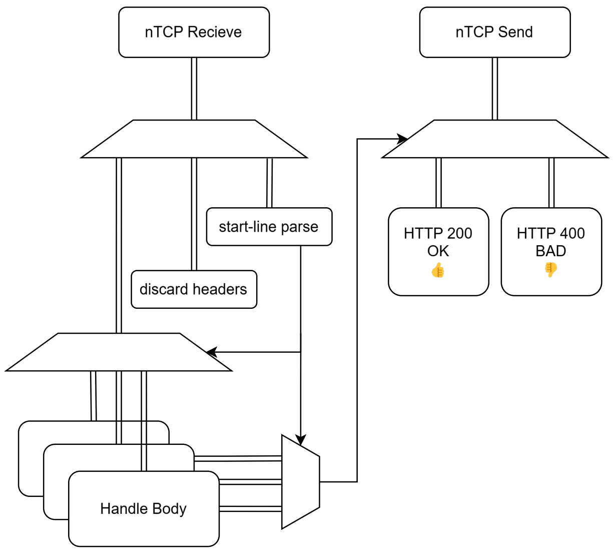


Figure 2: HTTP Request Parsing Workflow Diagram

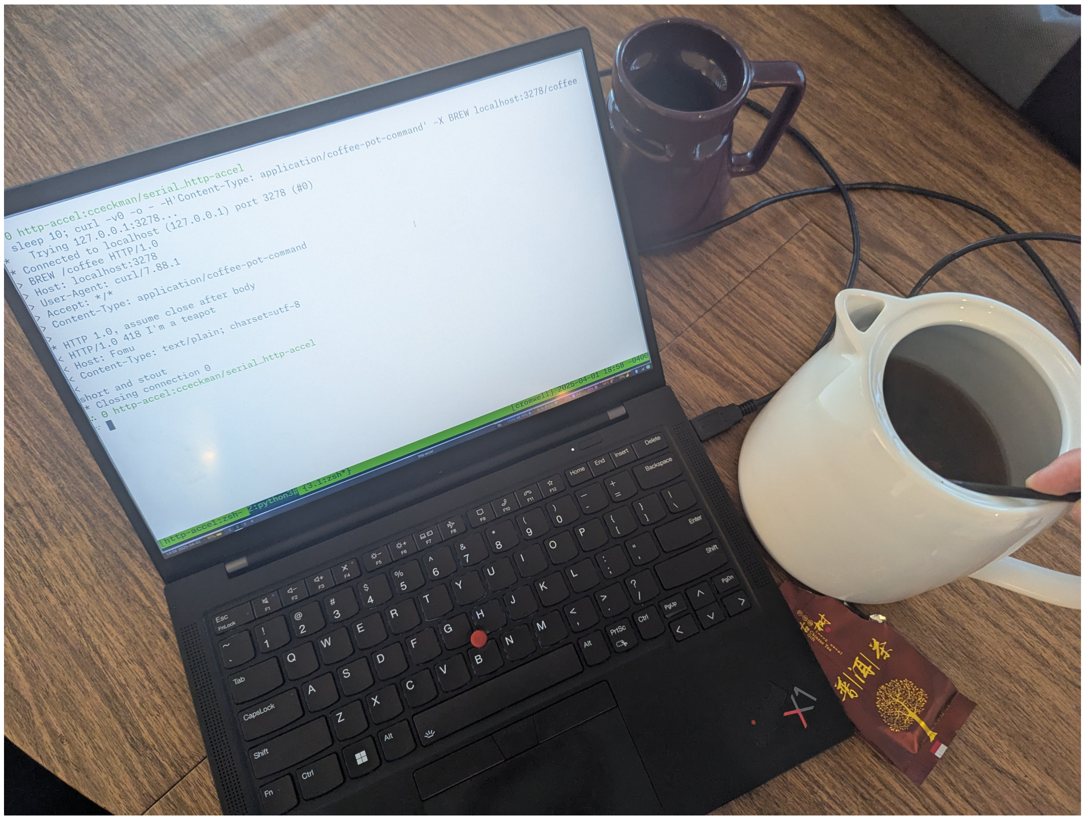
These documents define how an HTTP server should respond if it is a teapot.

This naturally leads to the question: What, exactly, *is* a teapot?

RFC2324 indicates that a teapot MAY have a body that is short and stout. RFC7168 further indicates that TEA capable pots are "expected to heat water through the use of electric elements".

Merriam-Webster defines a teapot as a vessel in which tea is brewed and served [10]. Previous implementers have put software HTCPCP servers for code 418 in a teapot [4], or have glued a commercial teapot on top of a laptop running an HTTP server [16]. These implementers follow RFC2324's recommendation of being short and stout, but they do not follow RFC7168's expectation to heat water through the use of an electric element. While hyperscalers have shown through their use of evaporative cooling that HTTP-serving hardware can be used to evaporate water (hence the term of art, cloud computing), this excessive degree of heating is counter to the purpose of the Hyper Text Coffee Pot Control Protocol (not to mention the Kyoto one).

As a more power-efficient and decentralized alternative, the authors added support for the `/coffee` path, and serve `HTTP 418 I'm a teapot` responses. The heat generated from serving these requests can be used to warm tea (note: energy transfer was minimal,



```
0 http-accel:cceckman/serial…http-accel
 sleep 10; curl -v0 -o - -H'Content-Type: application/coffee-pot-command' -X BREW localhost:3278/coffee
*   Trying 127.0.0.1:3278...
* Connected to localhost (127.0.0.1) port 3278 (#0)
> BREW /coffee HTTP/1.0
> Host: localhost:3278
> User-Agent: curl/7.88.1
> Accept: */*
> Content-Type: application/coffee-pot-command
>
* HTTP 1.0, assume close after body
< HTTP/1.0 418 I'm a teapot
< Host: Fomu
< Content-Type: text/plain; charset=utf-8
<
short and stout
* Closing connection 0
: 0 http-accel:cceckman/serial…http-accel
:
[http-accel:zsh-  2:python3#  {3.1:zsh*}          [cromwell] 2025-04-01 18:58 -0400
```

Figure 3: RFC2324 compatibility tested (and tasted)

but non-zero). Therefore, the authors have created special-purpose electronics that serve the dual purpose of responding to RFC2324 requests while heating tea, making this the first RFC7168-compliant HTTP 418 endpoint we're aware of.

4 Results

The resulting design uses a nice fraction of logic cells (3684 of the Fomu's 5280) and a small number of RAMs (5 of 30). Timing closes at 48MHz (USB logic) and 12MHz (main logic) with overhead to spare (51MHz and 21MHz, respectively).

5 Future work

See TODOs at https://github.com/cceckman/http-accel. In accordance with RFC 9759[1], we expect all outstanding work in this area to be completed within two weeks.

6 Conclusions

In conclusion, we managed to put an HTTP server on a Fomu FPGA platform using nTCP and other ETLAs.

Considering the industry's trend towards specialized hardware and AIASS (Artificial Intelligence AS a Service), developers of application accelerators should consider integrating the product's IP, TCP, HTTP, JSON parsing, APIs, storage, bot protection, authentication, A/B testing, billing, safety, and legal obligations into the hardware design.

Acknowledgments

Thanks to Q for supporting Charles while working on this. Thanks to M+T for leaving Stephen enough sleep to work on this. Thanks to Scout, Bin, Wolfgang, Ludwig, and Arya for their emotional support. And last, but not least, thank you Harry Q. Bovik for giving us a venue for this nonsense.

References

[1] 2025. Unified Time Scaling for Temporal Coordination Frameworks. RFC 9759. doi:10.17487/RFC9759

[2] Ronen Heled Steve Orrin Chaim Linhard, Amit Klein. 2006–. HTTP Request Smuggling. https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf Accessed March 27, 2025.

[3] Eric S. Raymond. [n. d.]. The Jargon File, version 4.4.6: blinkenlights. https://jargon-file.org/archive/jargon-4.4.6.dos.txt. Definition of "blinkenlights.

[4] Error 418. [n. d.]. Error 418: I'm a teapot. https://error418.net/

[5] Espressif. [n. d.]. ESP-IDF example: HTTP Server. https://github.com/esp-rs/std-training/tree/main/intro/http-server

[6] Great Scott Gadgets. [n. d.]. LUNA: Amaranth HDL framework for monitoring, hacking, and developing USB devices. https://github.com/greatscottgadgets/luna

[7] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. 2004. pytest 8.3. https://github.com/pytest-dev/pytest

[8] David MacIver, Zac Hatfield-Dodds, and Many Contributors. 2019. Hypothesis: A new approach to property-based testing. Journal of Open Source Software 4, 43 (21 11 2019), 1891. doi:10.21105/joss.01891

[9] Larry M Masinter. 1998. Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0). RFC 2324. doi:10.17487/RFC2324

[10] Merriam-Webster. [n. d.]. Abulia. In Merriam-Webster.com dictionary. https://www.merriam-webster.com/dictionary/teapot

[11] Tim 'mithro' Ansell. 2018–. Fomu: An Open Source FPGA for Everyone. https://tomu.im/fomu.html Accessed March 24, 2025.

[12] Jeffrey C Mogul. 2003. TCP offload is a dumb idea whose time has come. In 9th Workshop on Hot Topics in Operating Systems (HotOS IX). https://www.usenix.org/legacy/events/hotos03/tech/full_papers/mogul/mogul.pdfe

[13] Tom Murphy, VII. 2022. Harder Drive: Hard drives we didn't want or need. In A record of the proceedings of SIGBOVIK 2022 (Pittsburgh, USA). ACH, 259–277. tom7.org/harder.

[14] Imran Nazar. 2014. The Hyper Text Coffee Pot Control Protocol for Tea Efflux Appliances (HTCPCP-TEA). RFC 7168. doi:10.17487/RFC7168

[15] Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. 1996. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945. doi:10.17487/RFC1945

[16] Joe Reddington. [n. d.]. 418 Error Code Teapot. https://web.archive.org/web/20150906071854/http://joereddington.com/projects/418-error-code-teapot/ Archived at the Wayback Machine: September 6, 2015.

[17] Tiny Tapeout. [n. d.]. Tiny Tapeout. https://tinytapeout.com/

[18] Amaranth HDL Development Team. [n. d.]. Amaranth HDL: A modern hardware description language. https://github.com/amaranth-lang/amaranth