Kernel HTTPS/TCP/IP stack for HTTP DDoS mitigation

Alexander Krizhanovsky

Tempesta Technologies, Inc.

ak@tempesta-tech.com

Who am I?

- CEO & CTO at **Tempesta Technologies** *(Seattle, WA)*
- Developing **Tempesta FW** – open source Linux *Application Delivery Controller (ADC)*
- **Custom software development** in:
  - *high performance network traffic processing* e.g. **WAF** mentioned in **Gartner magic quadrant**
  - *Databases* e.g. **MariaDB SQL System Versioning** *https://github.com/tempesta-tech/mariadb* https://m17.mariadb.com/session/technical-preview-temporal-querying-asof

HTTPS challenges

- HTTP(S) is a core protocol for the Internet (IoT, SaaS, Social networks etc.)
- HTTP(S) DDoS is tricky
  - Asymmetric DDoS (compression, TLS handshake etc.)
  - A lot of IP addresses with low traffic
  - Machine learning is used for clustering
  - How to filter out all HTTP requests with `"Host: www.example.com:80"`?
  - "Lessons From Defending The Indefensible": https://www.youtube.com/watch?v=pCVTEx1ouyk

Tempesta Technologies

TCP stream filter

- **IPtables strings**, **BPF, XDP, NIC filters**
  - HTTP headers can cross packet bounds
  - Scan large URI or Cookie for Host value?
- **Web accelerator**
  - aren’t designed (suitable) for HTTP filtering

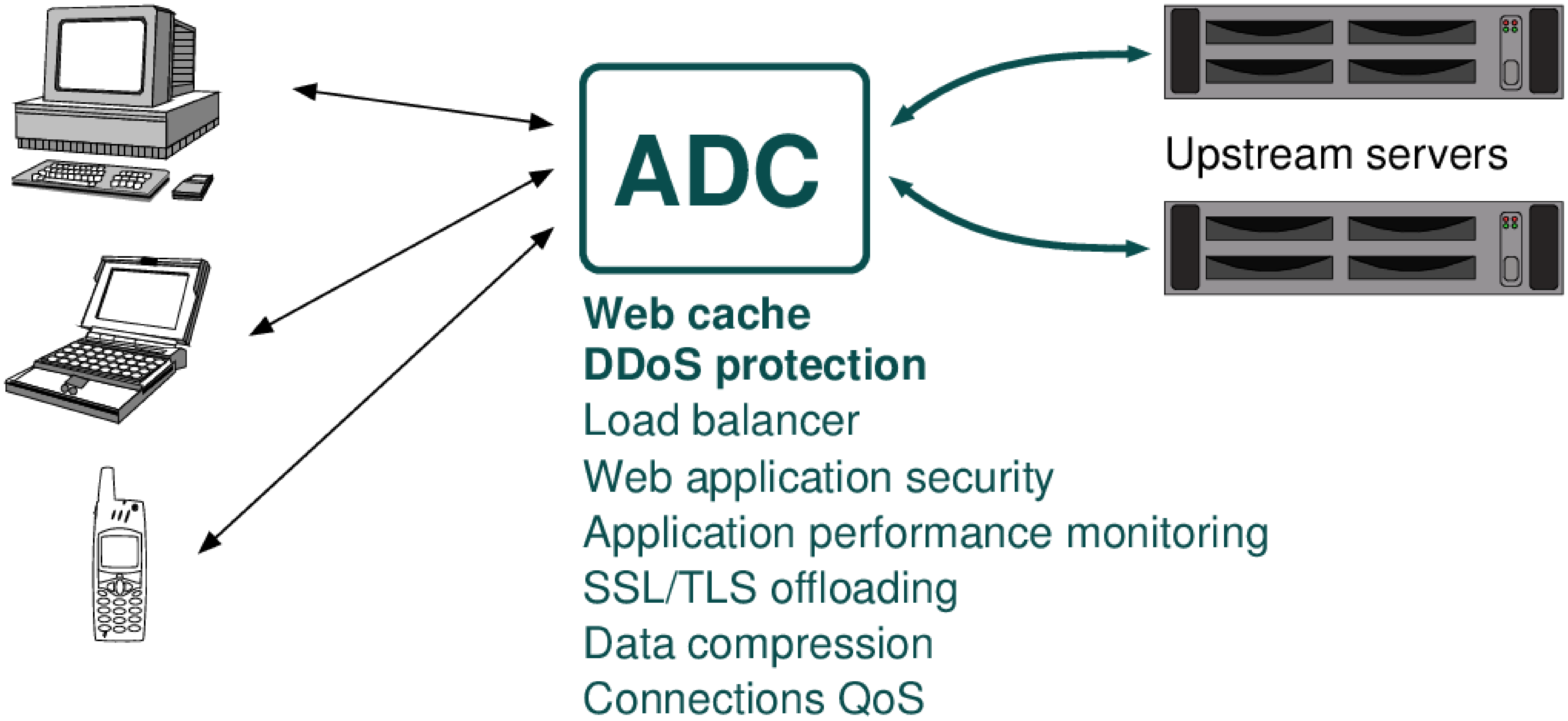
IPS vs HTTP DDoS

- e.g. Suricata, has powerful rules syntax at L3-L7
- Not a TCP end point => evasions are possible
- SSL/TLS
  - SSL terminator is required => many data copies & context switches
  - or double SSL processing (at IDS & at Web server)
- Double HTTP parsing
- Doesn’t improve Web server peroformance (mitigation != prevention)

Interbreed an HTTP accelerator and a firewall

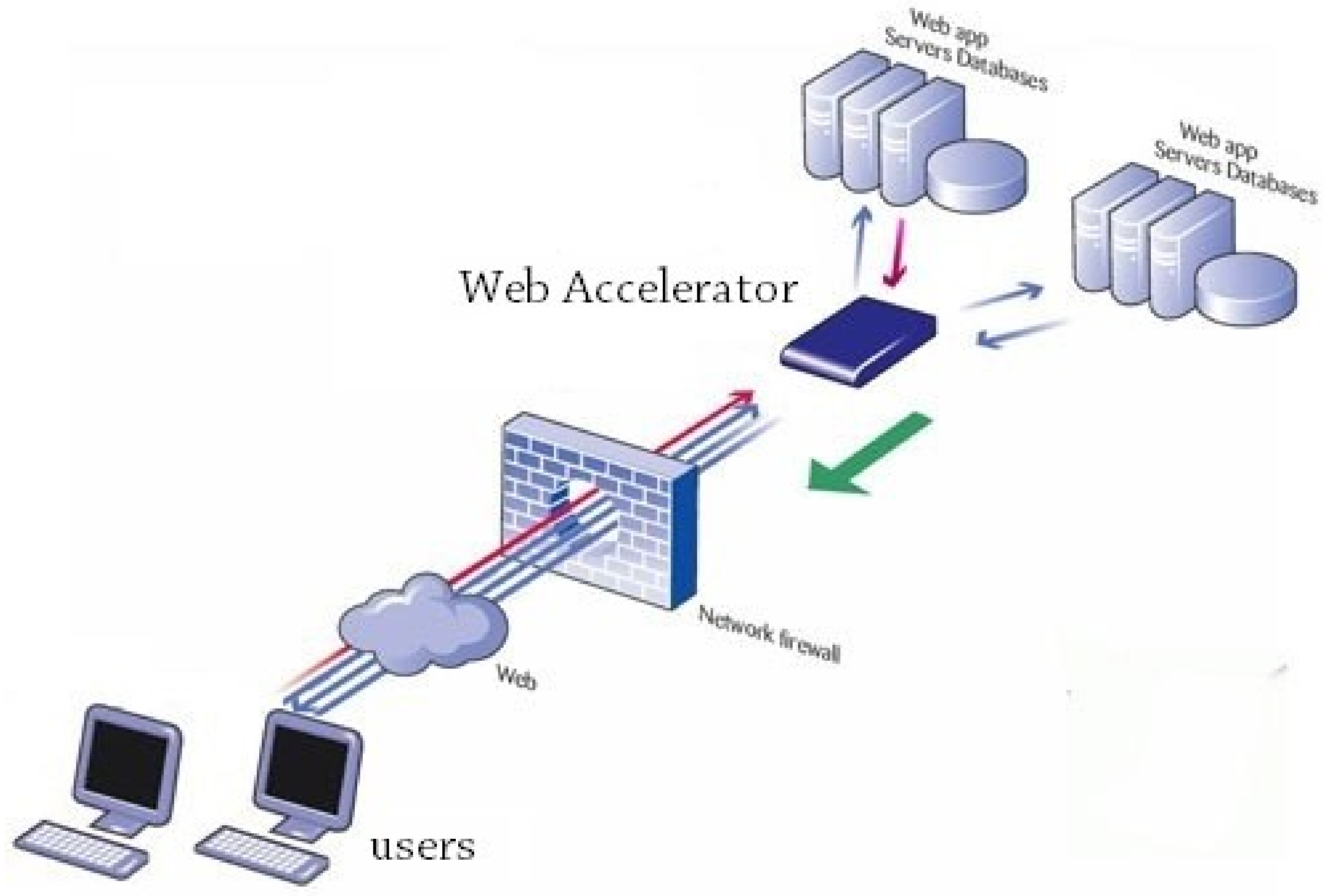
- TCP & TLS end point
- Very **fast HTTP parser** to process HTTP floods
- Network I/O optimized for **massive ingress traffic**
- **Advanced filtering** abilities at all network layers
- Very fast Web cache to mitigate DDoS which we can’t filter out
  - ML takes some time for bots clusterization
  - False negatives are unavoidable

Application Delivery Controller (ADC)



WAF accelerator

- **Just like Web accelerator**
- **Advanced load balancing:**
  - Server groups by any HTTP field
  - Rendezvous hashing
  - Ratio
  - Adaptive & predictive
- Some DDoS attacks can be **just normally serviced**



Tempesta Technologies

Application layer DDoS

	Service from Cache	Rate limit
Nginx	22us	23us

- *(Additional logic in limiting module)*
- **Fail2Ban**: write to the log, parse the log, write to the log, parse the log…

Tempesta Technologies

Application layer DDoS

	Service from Cache	Rate limit
Nginx	22us	23us

- *(Additional logic in limiting module)*
- **Fail2Ban**: write to the *log*, parse the *log*, write to the *log*, parse the *log*… **- really in 21th century?!**
- **tight integration** of Web accelerator and a firewall is needed

Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering etc.

Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, **filtering?** etc.

Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, **filtering?** etc.
  - **C10K**

Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, **filtering?** etc.
  - **C10K –** *is it a problem for bot-net?* **SSL?**
  - what about tons of `'GET / HTTP/1.0\n\n'`?

CORNER CASES!

Tempesta Technologies

Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, **filtering?** etc.
  - **C10K –** *is it a problem for bot-net?* **SSL?**
  - what about tons of `'GET / HTTP/1.0\n\n'`?

CORNER CASES!

- **Kernel-mode Web-accelerators**: TUX, kHTTPd
  - basically the same sockets and threads
  - zero-copy → *sendfile(), lazy TLB*

Tempesta Technologies

Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, **filtering?** etc.
  - **C10K –** *is it a problem for bot-net?* **SSL?**
  - what about tons of `'GET / HTTP/1.0\n\n'`?

CORNER CASES!

- **Kernel-mode Web-accelerators**: TUX, kHTTPd
  - basically the same sockets and threads
  - zero-copy → *sendfile(), lazy TLB* => not needed

Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, **filtering?** etc.
  - **C10K –** *is it a problem for bot-net?* **SSL?**
  - what about tons of `'GET / HTTP/1.0\n\n'`?
- **Kernel-mode Web-accelerators**: TUX, kHTTPd
  - basically the same sockets and threads
  - zero-copy → *sendfile(), lazy TLB* => not needed

CORNER CASES! NEED AGAIN TO MITIGATE HTTPS DDOS

Tempesta Technologies

Web-accelerators are slow: SSL/TLS copying

- User-kernel space copying
  - Copy network data to user space
  - Encrypt/decrypt it
  - Copy the date to kernel for transmission (or **`splice()`**)
- **Kernel-mode TLS**
  - Facebook & RedHat: *https://lwn.net/Articles/666509/*
  - Mellanox: https://netdevconf.org/1.2/session.html?boris-pismenny
  - Netflix: *https://people.freebsd.org/~rrs/asiabsd_2015_tls.pdf*

Tempesta Technologies

Linux kernel TLS (since 4.13)

- `CONFIG_TLS` (switched off by default)
- Symmetric encryption only (no handshake)
- Example (*https://github.com/Mellanox/tls-af_ktls_tool*):

```
struct tls12_crypto_info_aes_gcm_128 ci = {
    .version = TLS_1_2_VERSION, .cipher_type = TLS_CIPHER_AES_GCM_128 };
connect(sd, ..., ...);
gnutls_handshake(*session);
gnutls_record_get_state(session, ..., ..., iv, key, seq);
memcpy(ci.iv, seq, TLS_CIPHER_AES_GCM_128_IV_SIZE);
memcpy(ci.rec_seq, seq, TLS_CIPHER_AES_GCM_128_REC_SEQ_SIZE);
memcpy(ci.key, key, TLS_CIPHER_AES_GCM_128_KEY_SIZE);
memcpy(ci.salt, iv, TLS_CIPHER_AES_GCM_128_SALT_SIZE);
setsockopt(sd, SOL_TCP, TCP_ULP, "tls", sizeof("tls"));
setsockopt(sd, SOL_TLS, TLS_TX, &ci, sizeof(ci));
```

Tempesta Technologies

Linux kernel TLS & DDoS

- Most Facebook users have established sessions
- **TLS handshake is still an issue**
  - TLS 1.3 has 1-RTT handshake and is almost here
  - TLS 1.2 must live for a long time (*is Windows XP still alive?*)
  - TLS renegotiation

Tempesta Technologies

Web-accelerators are slow: profile

%	symbol name
1.5719	ngx_http_parse_header_line
1.0303	ngx_vslprintf
0.6401	memcpy
0.5807	recv
0.5156	ngx_linux_sendfile_chain
0.4990	ngx_http_limit_req_handler

=> flat profile

Web-accelerators are slow: syscalls

```
epoll_wait(.., {{EPOLLIN, ....}},...)
recvfrom(3, "GET / HTTP/1.1\r\nHost:...", ...)
write(1, “...limiting requests, excess...", ...)
writev(3, "HTTP/1.1 503 Service...", ...)
sendfile(3,..., 383)
recvfrom(3, ...) = -1 EAGAIN
epoll_wait(.., {{EPOLLIN, ....}}, ...)
recvfrom(3, "", 1024, 0, NULL, NULL) = 0
close(3)
```

Tempesta Technologies

Web-accelerators are slow: HTTP parser

Start: *state = 1, *str_ptr = 'b'*

```
while (++str_ptr) {
    switch (state) { <= check state
    case 1:
        switch (*str_ptr) {
        case 'a':
            ...
            state = 1
        case 'b':
            ...
            state = 2
        }
    case 2:
        ...
    }
    ...
}
```

Web-accelerators are slow: HTTP parser

Start: *state = 1, *str_ptr = 'b'*

```
while (++str_ptr) {
    switch (state) {
    case 1:
        switch (*str_ptr) {
        case 'a':
            ...
            state = 1
        case 'b':
            ...
            state = 2 <= set state
        }
    case 2:
        ...
    }
    ...
}
```

Web-accelerators are slow: HTTP parser

Start: *state = 1, *str_ptr = 'b'*

```
while (++str_ptr) {
    switch (state) {
    case 1:
        switch (*str_ptr) {
        case 'a':
            ...
            state = 1
        case 'b':
            ...
            state = 2
        }
    case 2:
        ...
    }
    ... <= jump to while
}
```

Web-accelerators are slow: HTTP parser

Start: *state = 1, *str_ptr = 'b'*

```
while (++str_ptr) {
    switch (state) { <= check state
    case 1:
        switch (*str_ptr) {
        case 'a':
            ...
            state = 1
        case 'b':
            ...
            state = 2
        }
    case 2:
        ...
    }
    ...
}
```

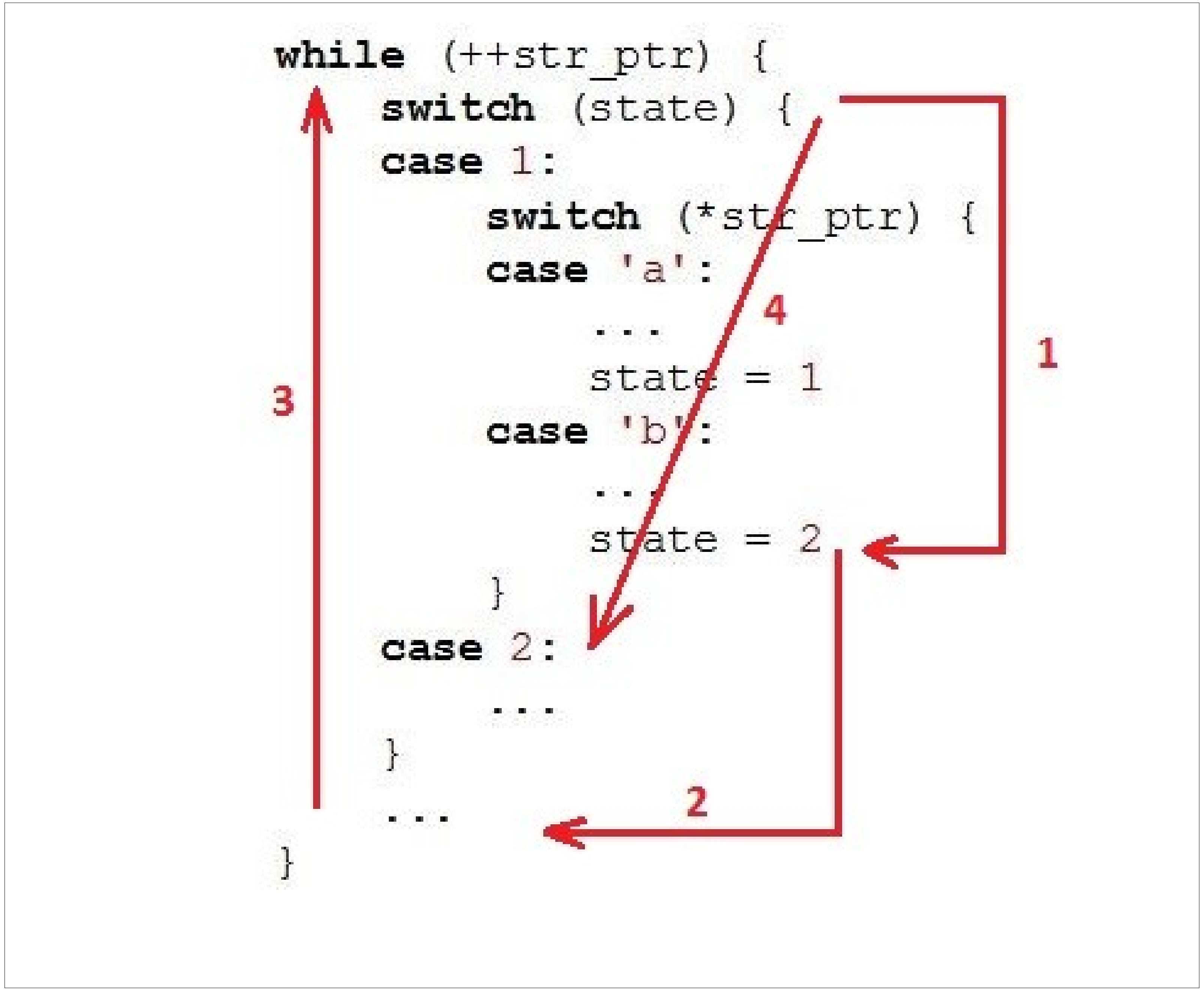
Web-accelerators are slow: HTTP parser

Start: *state = 1, *str_ptr = 'b'*

```
while (++str_ptr) {
    switch (state) {
    case 1:
        switch (*str_ptr) {
        case 'a':
            ...
            state = 1
        case 'b':
            ...
            state = 2
        }
    case 2:
        ...   <= do something
    }
    ...
}
```

Tempesta Technologies

Web-accelerators are slow: HTTP parser




```
while (1):
STATE_1:
    switch (*str_ptr) {
    case 'a':
        ...
        ++str_ptr
        goto STATE_1
    case 'b':
        ...
        ++str_ptr
STATE_2:
    ...
```

Web-accelerators are slow: strings

- We have AVX2, but GLIBC doesn’t still use it
- HTTP strings are special:
  - No `‘\0’`-termination (if you’re zero-copy)
  - Special delimiters (`‘:’` or `CRLF`)
  - `strcasecmp()`: no need case conversion for one string
  - `strspn()`: limited number of accepted alphabets
- `switch()`-driven FSM is even worse

Fast HTTP parser

- *http://natsys-lab.blogspot.ru/2014/11/the-fast-finite-state-machine-for-http.html*
  - **1.6-1.8 times faster than Nginx’s**
- HTTP optimized AVX2 strings processing: *http://natsys-lab.blogspot.ru/2016/10/http-strings-processing-using-c-sse42.html*
  - *~1KB strings:*
  - `strncasecmp()` **~x3 faster** than GLIBC’s
  - URI matching **~x6 faster** than GLIBC’s `strspn()`
  - `kernel_fpu_begin()`/`kernel_fpu_end()` for whole softirq shot

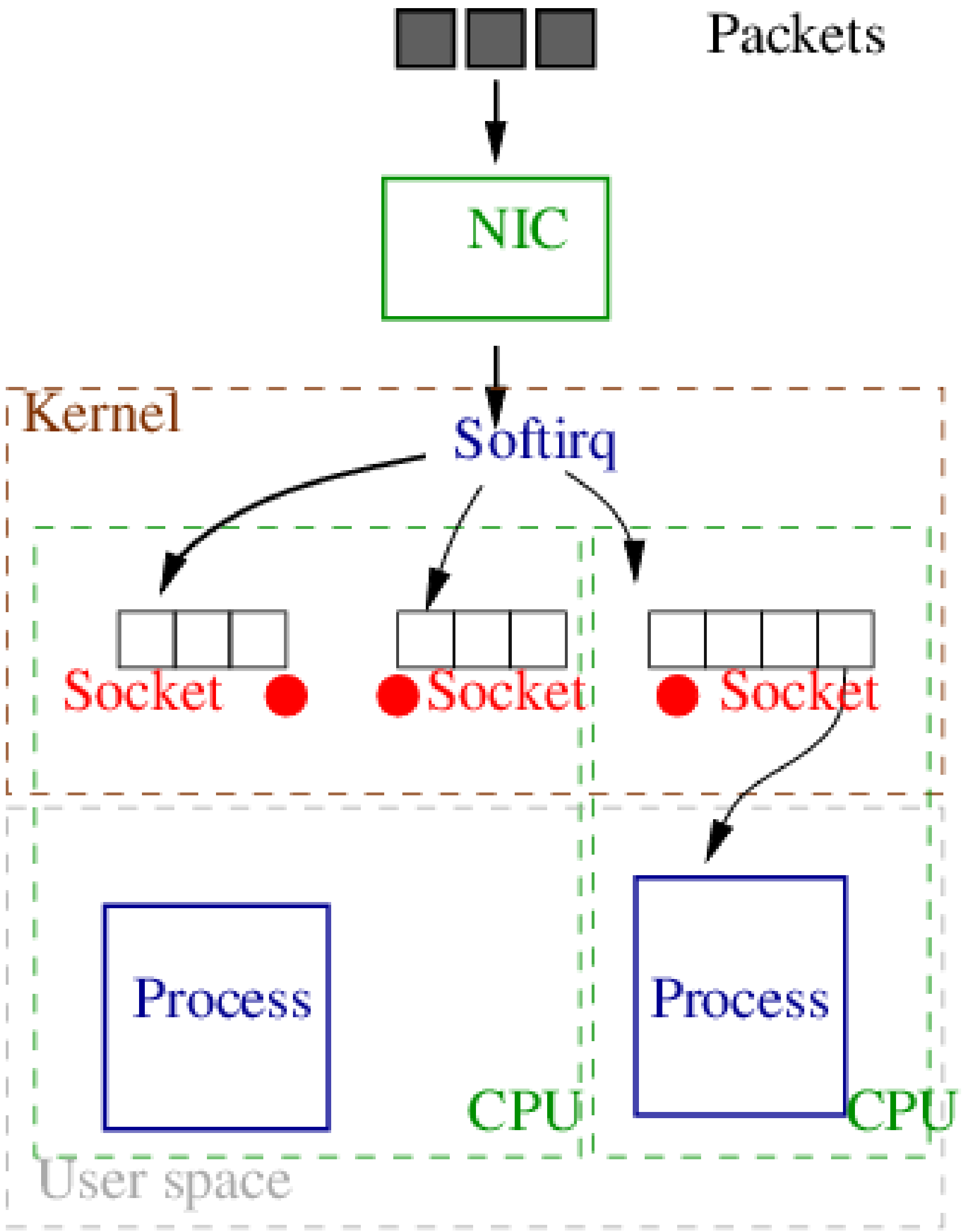
Tempesta Technologies

HTTP strong validation

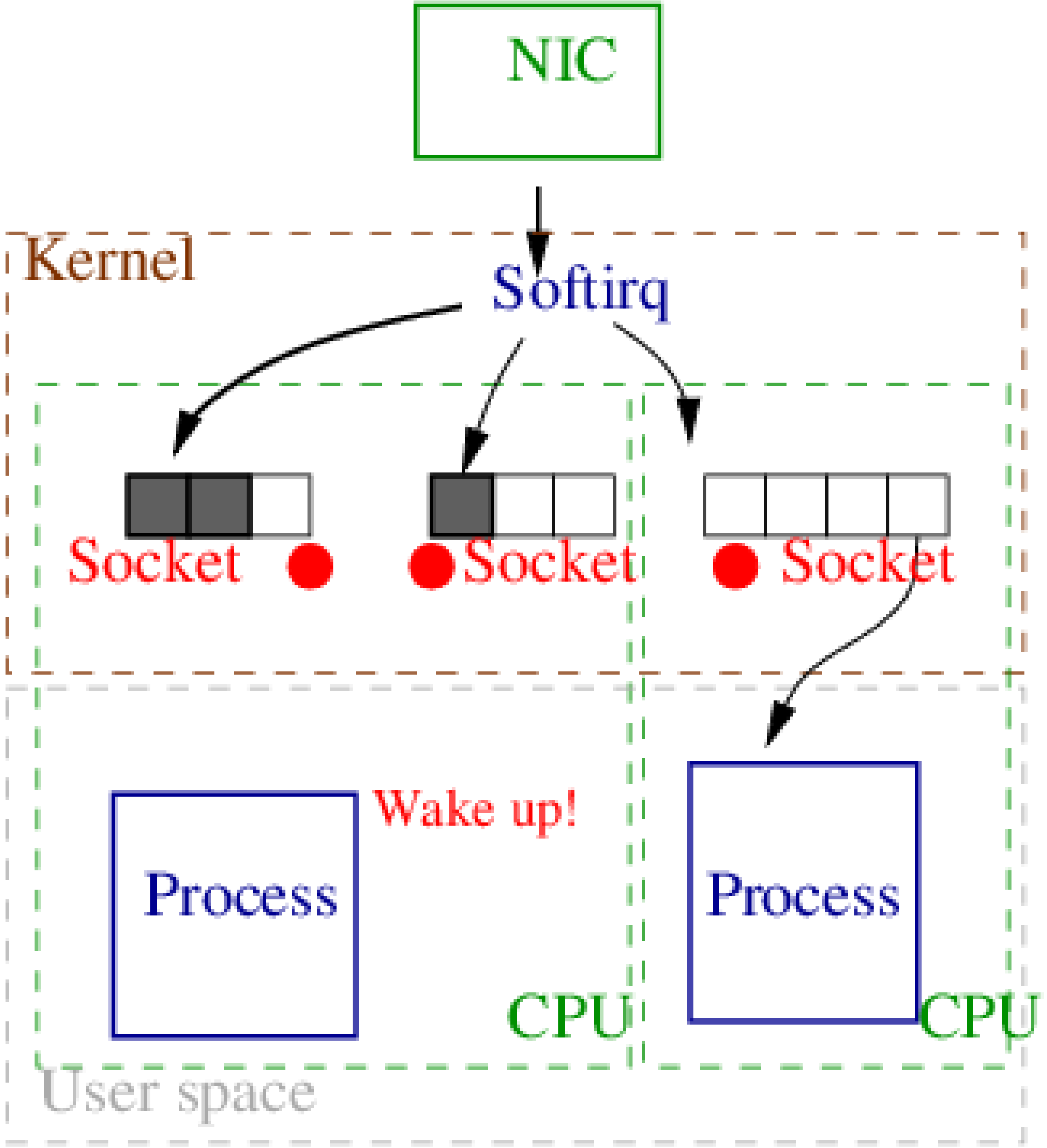
TODO: *https://github.com/tempesta-tech/tempesta/issues/628*

- **Injections:** specify allowed URI characters for a Web app
- Resistant to large HTTP fields

Web-accelerators are slow: async I/O

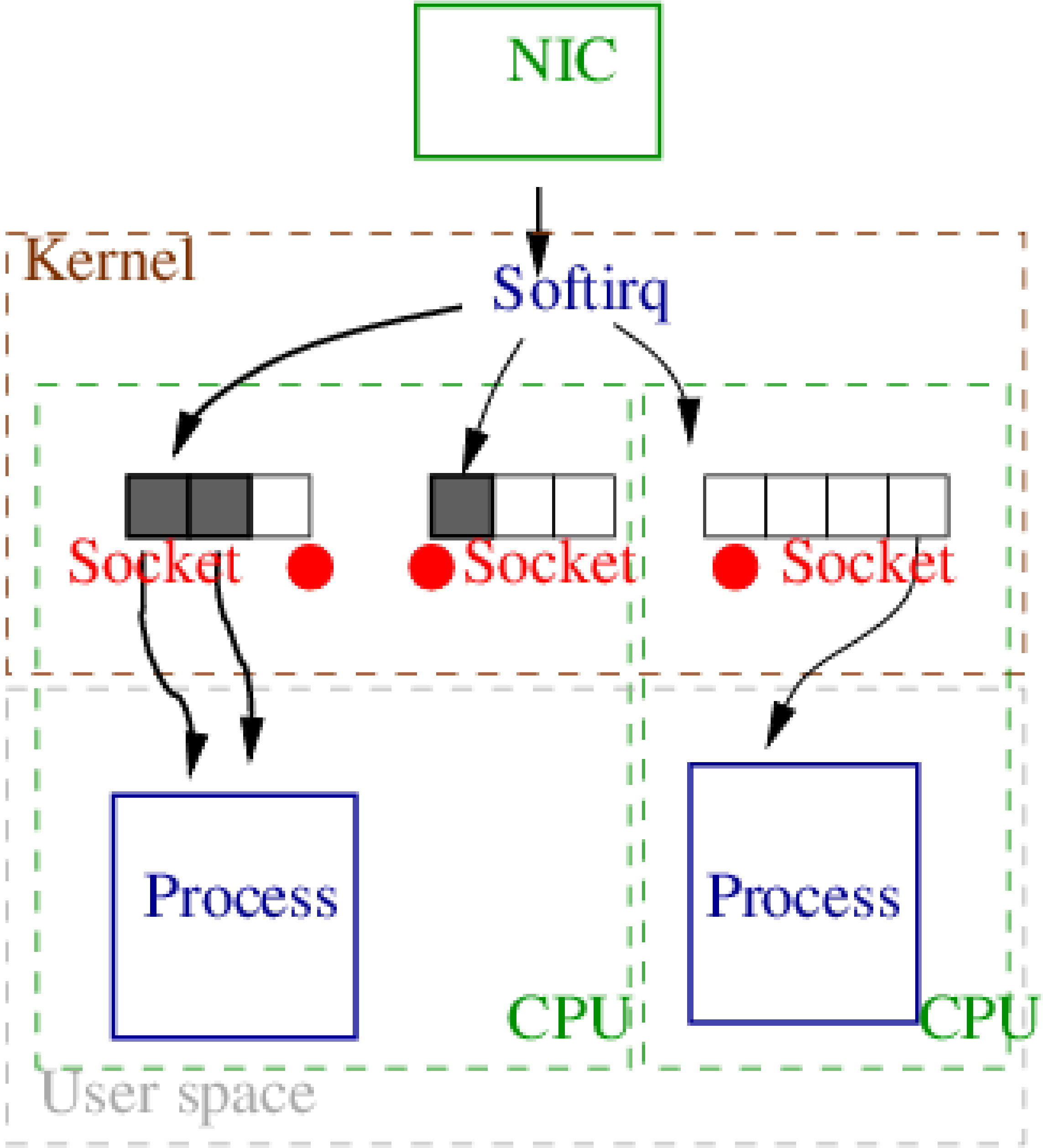


Web-accelerators are slow: async I/O



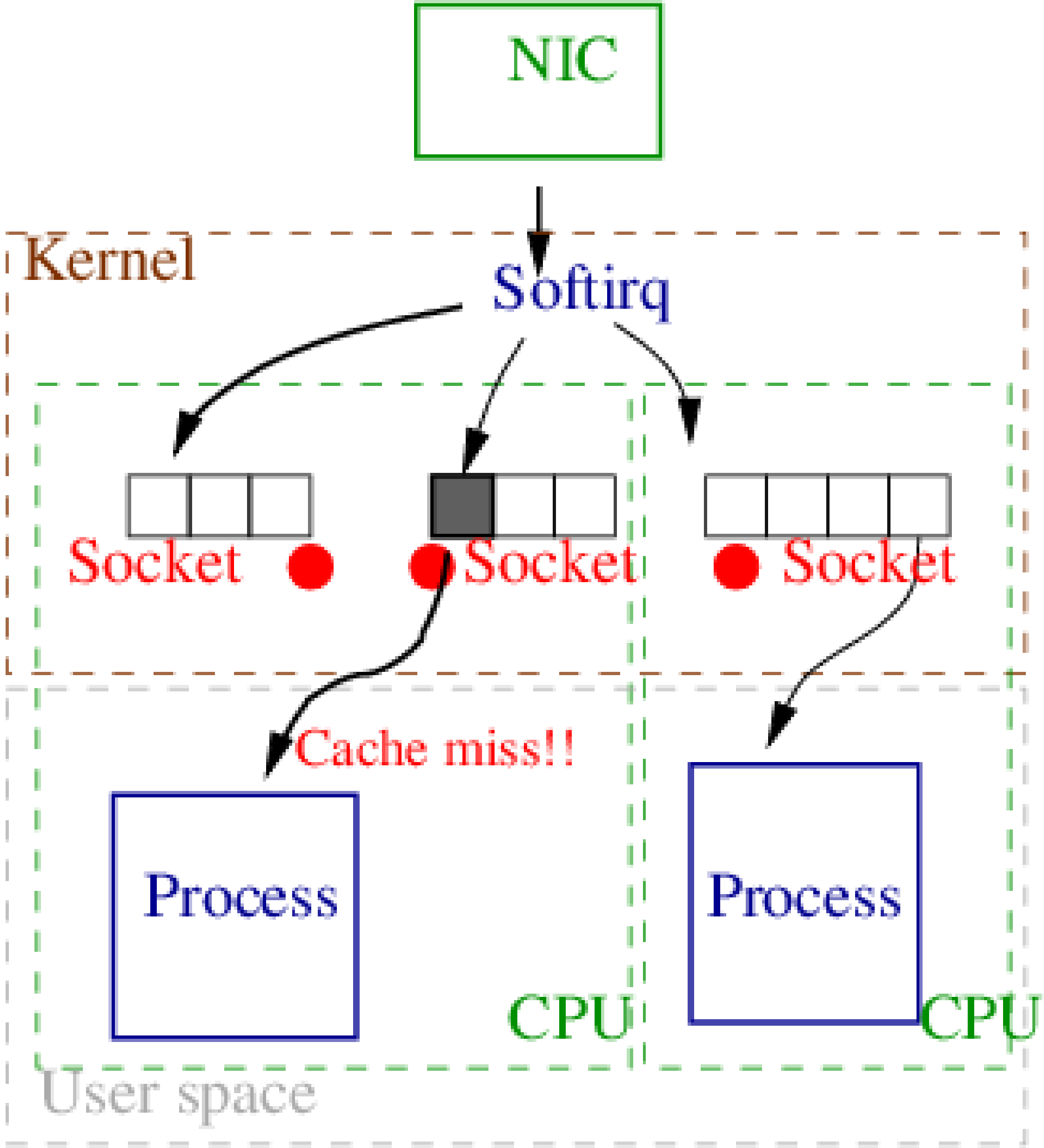
Tempesta Technologies

Web-accelerators are slow: async I/O



Web-accelerators are slow: async I/O

Web cache also resides In CPU caches and evicts requests



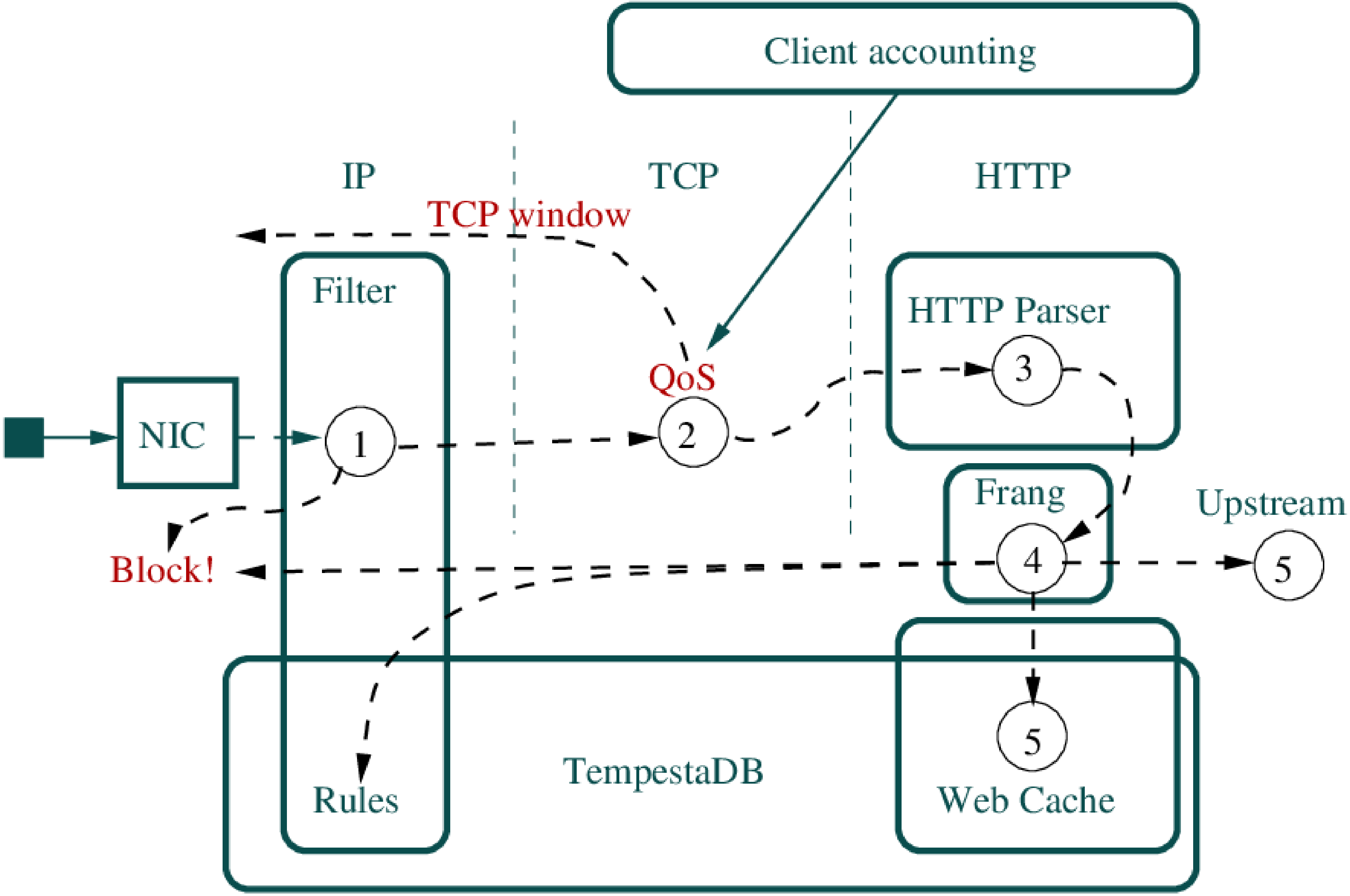
Tempesta Technologies

HTTPS/TCP/IP stack

- **Alternative to user space TCP/IP stacks**
- HTTPS is built into TCP/IP stack
- **Kernel TLS** (fork from mbedTLS) – no copying
  (**1 human month** to port TLS to kernel!)
- HTTP firewall plus to IPtables and Socket filter
- Very f**ast HTTP parser** and strings processing using AVX2
- **Cache-conscious** in-memory Web-cache for DDoS mitigation
- TODO
  - **HTTP QoS** for asymmetric DDoS mitigation
  - **DSL** for multi-layer filter rules

Tempesta Technologies

Tempesta FW



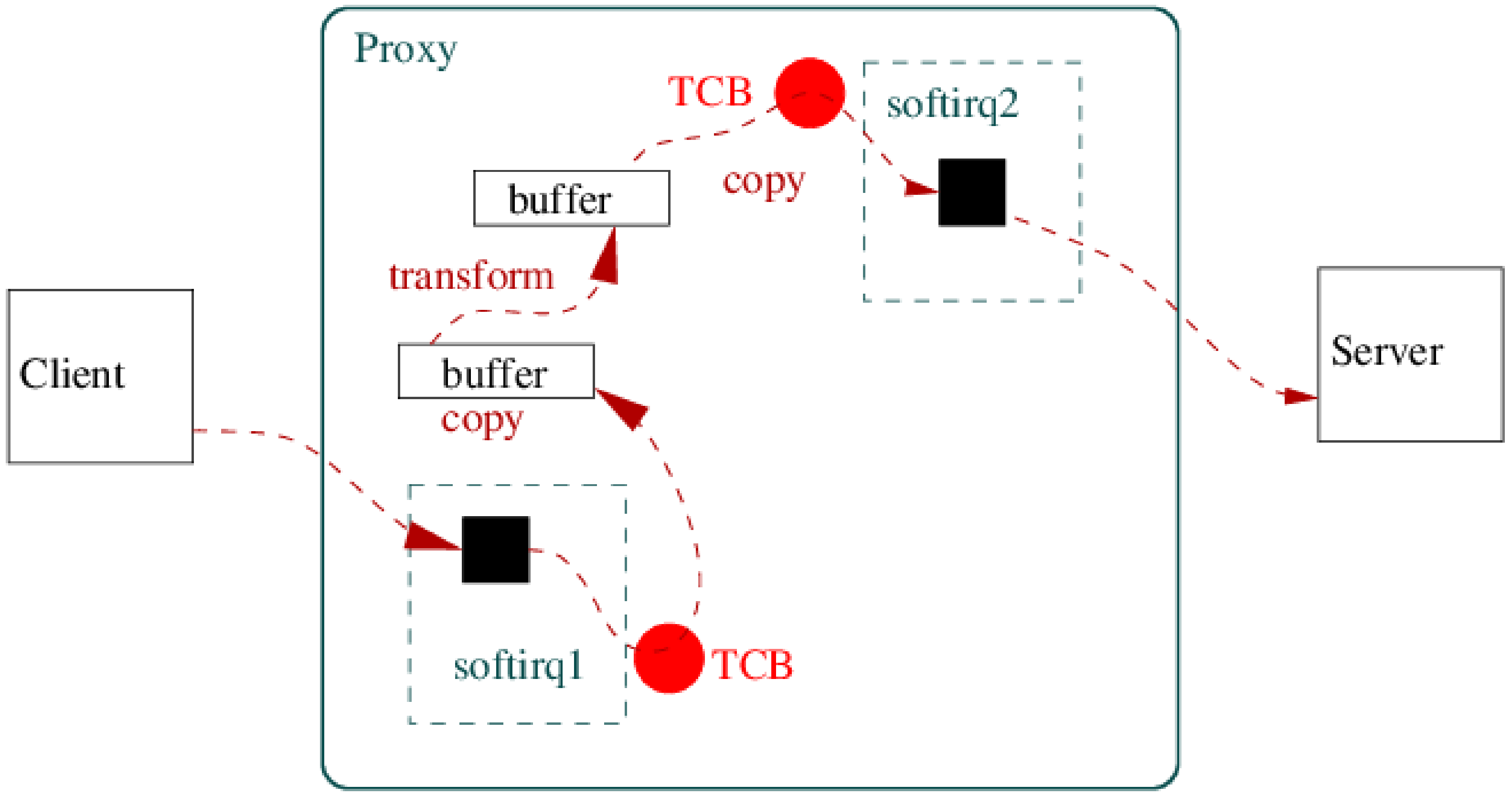
Client accounting
IP
TCP
HTTP
TCP window
Filter
HTTP Parser
QoS
NIC
1
2
3
Frang
Upstream
Block!
4
5
5
Rules
TempestaDB
Web Cache

TODO: HTTP QoS for asymmetric DDoS mitigation

- *https://github.com/tempesta-tech/tempesta/issues/488*
- “Web2K: Bringing QoS to Web Servers” by Preeti Bhoj et al.
- **Local stress**: packet drops, queues overrun, response latency etc *(**kernel**: cheap statistics for asymmetric DDoS)*
- **Upsream stress**: `req_num / resp_num`, response time etc.
- **Static QoS rules** per vhost: HTTP RPS, integration w/ Qdisc - TBD
- Actions: reduce TCP window, don’t accept new connections, close existing connections

User space HTTP proxying

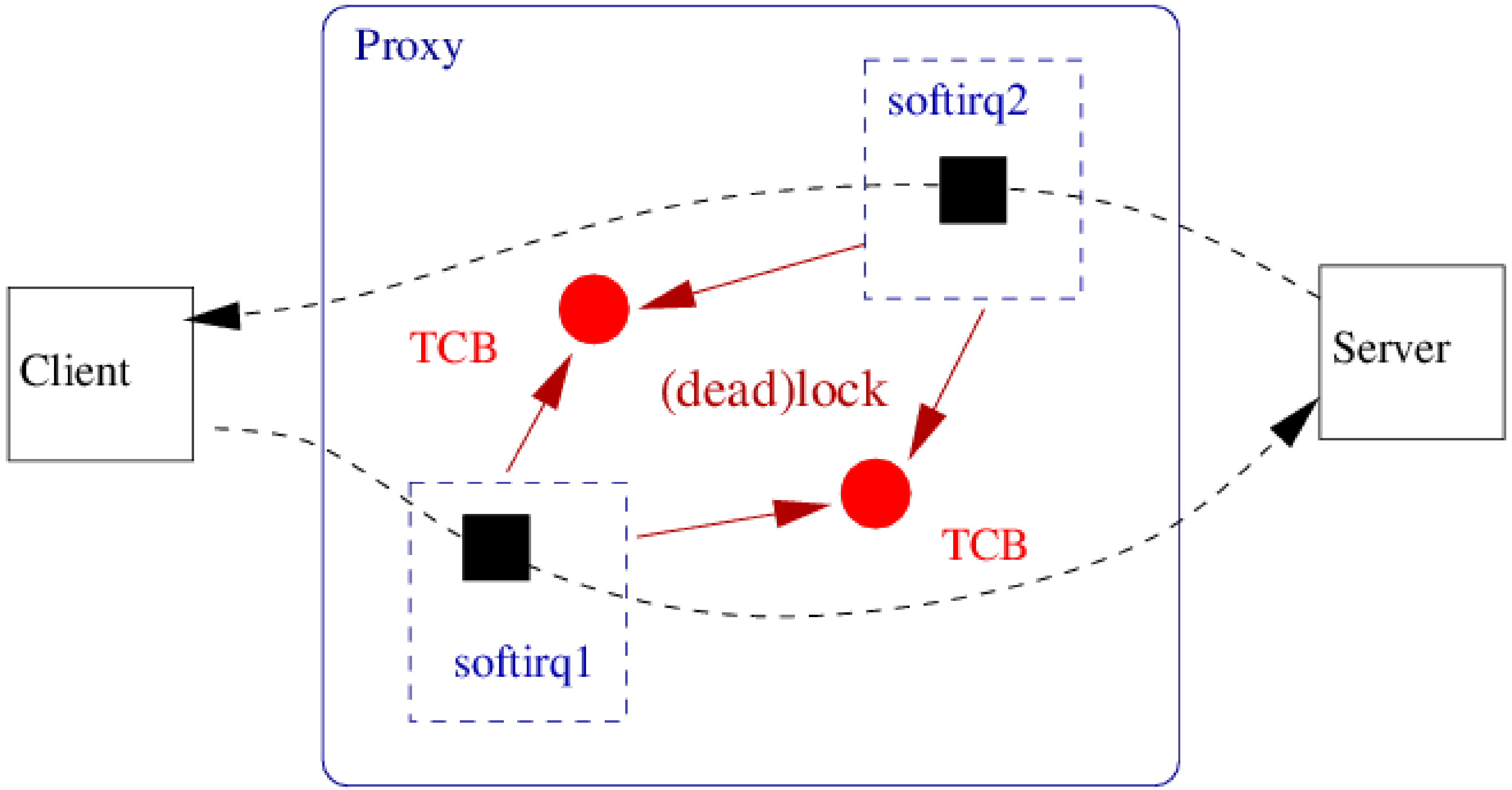
1. Receive request at CPU1
2. Copy request to user space
3. Update headers
4. Copy request to kernel space
5. Send the request from CPU2



- **3 data copies**
- Access TCP control blocks and data buffers from **different CPUs**

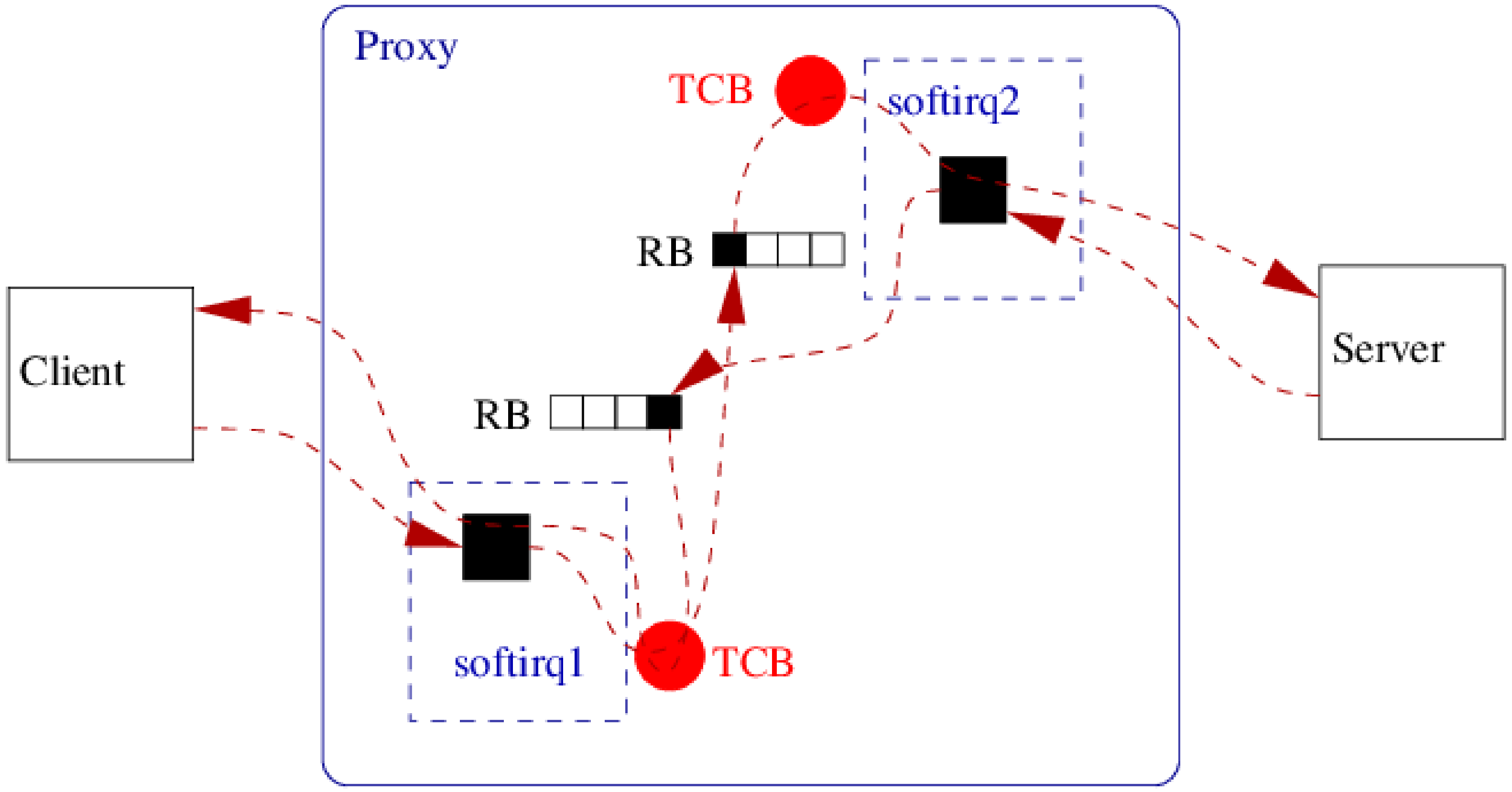
Synchronous sockets: HTTPS/TCP/IP stack

- Socket callbacks call TLS and HTTP processing
- Everything is processing in softirq (while the data is hot)
- No receive & accept queues
- No file descriptors
- Less locking



Synchronous sockets: HTTPS/TCP/IP stack

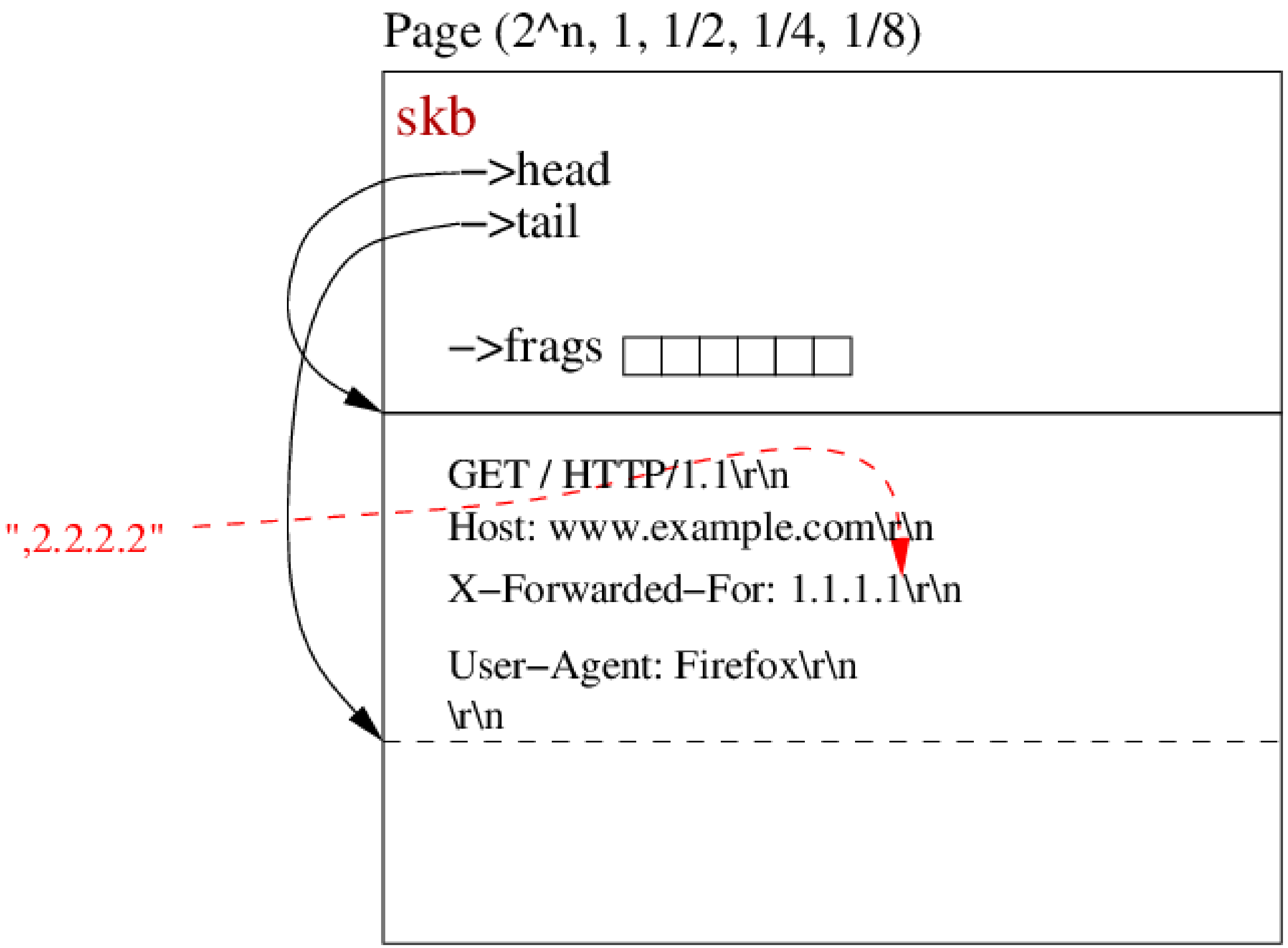
- Socket callbacks call TLS and HTTP processing
- Everything is processing in softirq (while the data is hot)
- No receive & accept queues
- No file descriptors
- Less locking
- Lock-free inter-CPU transport
- **=> faster socket reading**
- **=> lower latency**



Tempesta Technologies

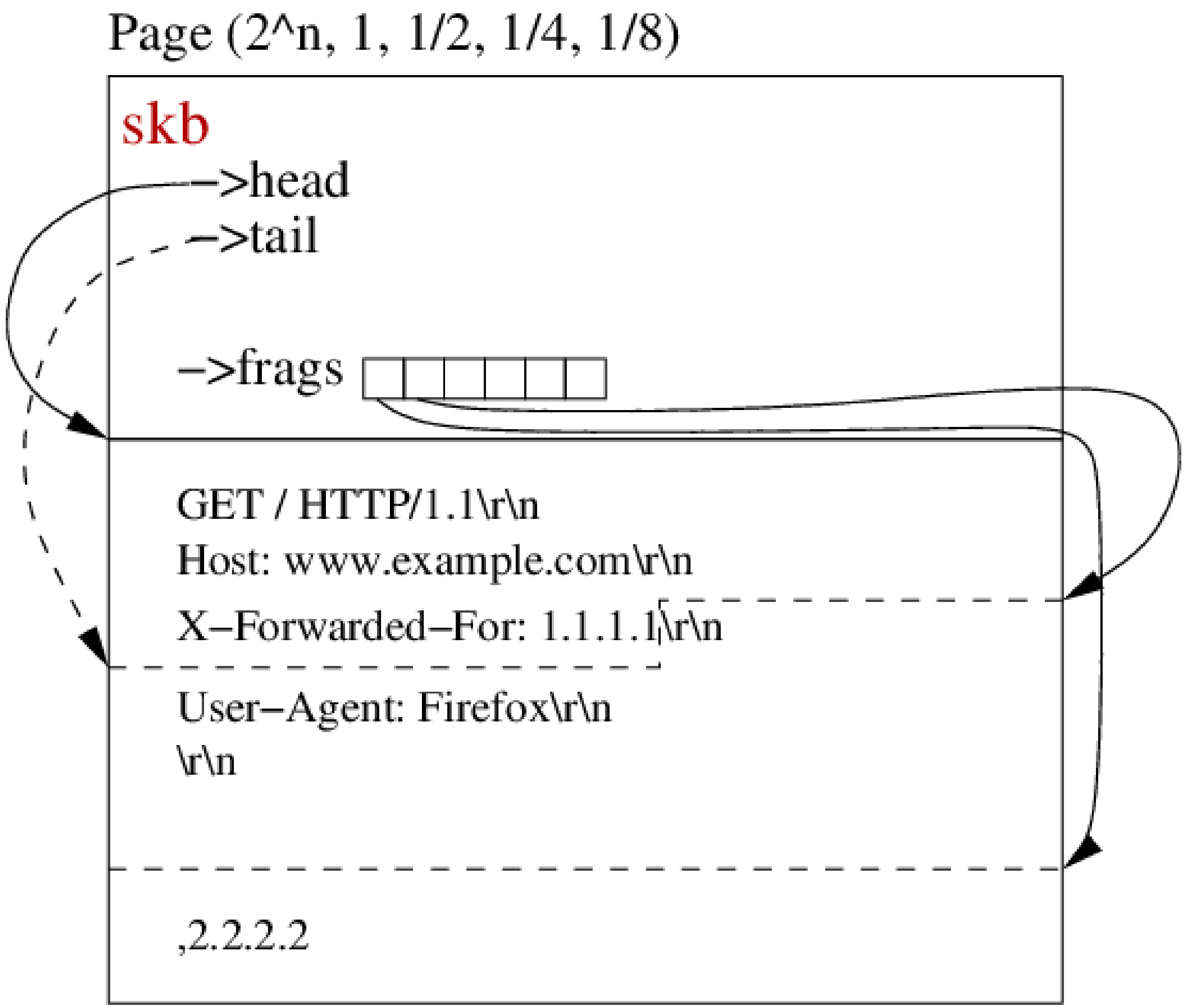
skb page allocator: zero-copy HTTP messages adjustment

- Add/remove/update HTTP headers w/o copies
- `skb` and its `head` are allocated in the same page fragment or a compound page



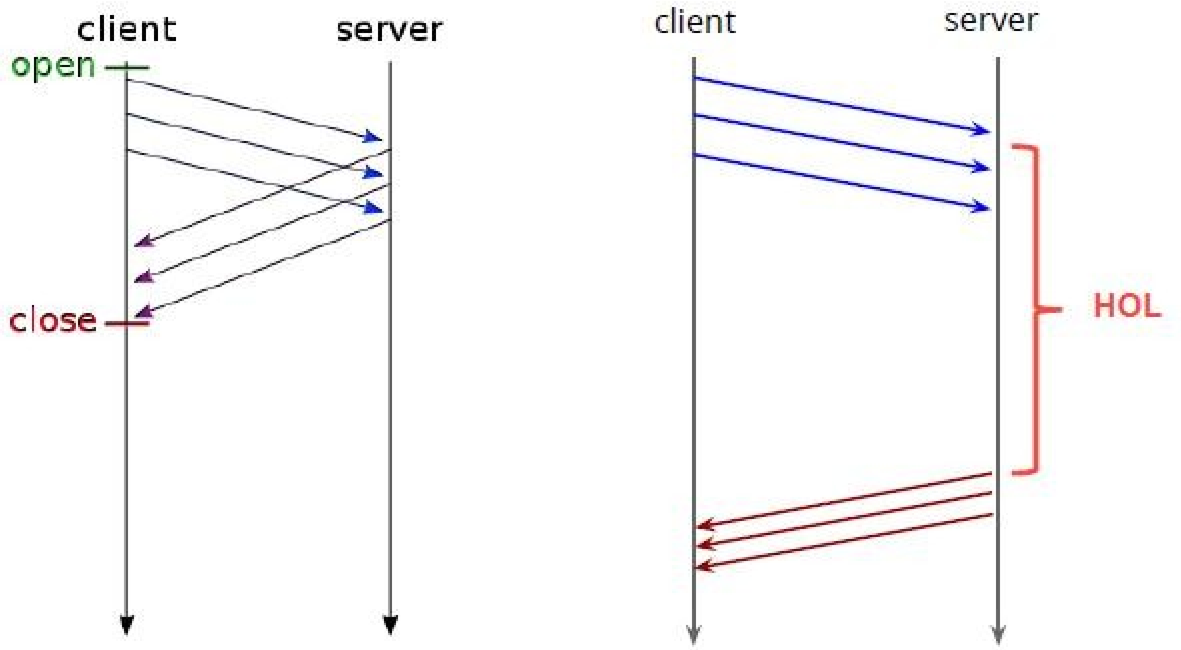
skb page allocator: zero-copy HTTP messages adjustment

- Add/remove/update HTTP headers w/o copies
- `skb` and its `head` are allocated in the same page fragment or a compound page



HTTP/2

- **Pros**
  - Responses are sent in *parallel* and in *any order* (no head-of-line blocking)
  - Compression
- **Cons**
  - Zero copy techniques aren’t applicable



=> For client connections (slow network), **not for LAN** (fast network)

QUIC?

- UDP-based with flow control
- 10% duplicates
- 0-RTT handshakes
- Implemented as a user-space library
- **Questions:**
  - Opaque UDP traffic just like UDP flood
  - TCP fast open + TLS 1.3 seem solve handshake problem

Frang: HTTP DoS

- **Rate limits**
  - request_rate, request_burst
  - connection_rate, connection_burst
  - concurrent_connections
  - TODO: tls handshakes
- **Slow HTTP**
  - client_header_timeout, client_body_timeout
  - http_header_cnt
  - http_header_chunk_cnt, http_body_chunk_cnt

Tempesta Technologies

Frang: WAF

- **Length limits:** http_uri_len, http_field_len, http_body_len
- **Content validation:** http_host_required, http_ct_required, http_ct_vals, http_methods
- **HTTP Response Splitting:** count and match requests and responses
- **Injections:** carefully verify allowed character sets
- ...and many upcoming filters: *https://github.com/tempesta-tech/tempesta/labels/security*
- **Not a featureful WAF**

Tempesta Technologies

Sticky cookie

- User/session identification
  - Cookie challenge for dummy DDoS bots
  - Persistent/sessions scheduling (no rescheduling on a server failure)
- **Enforce:** HTTP 302 redirect

```
sticky name=__tfw_user_id__ enforce;
```

Sticky cookie

- User/session identification
  - Cookie challenge for dummy DDoS bots
  - Persistent/sessions scheduling (no rescheduling on a server failure)
- **Enforce:** HTTP 302 redirect

```
sticky name=__tfw_user_id__ enforce;
```

- TODO: **JavaScript challenge**
  https://github.com/tempesta-tech/tempesta/issues/536

Tempesta Technologies

TODO: Tempesta language

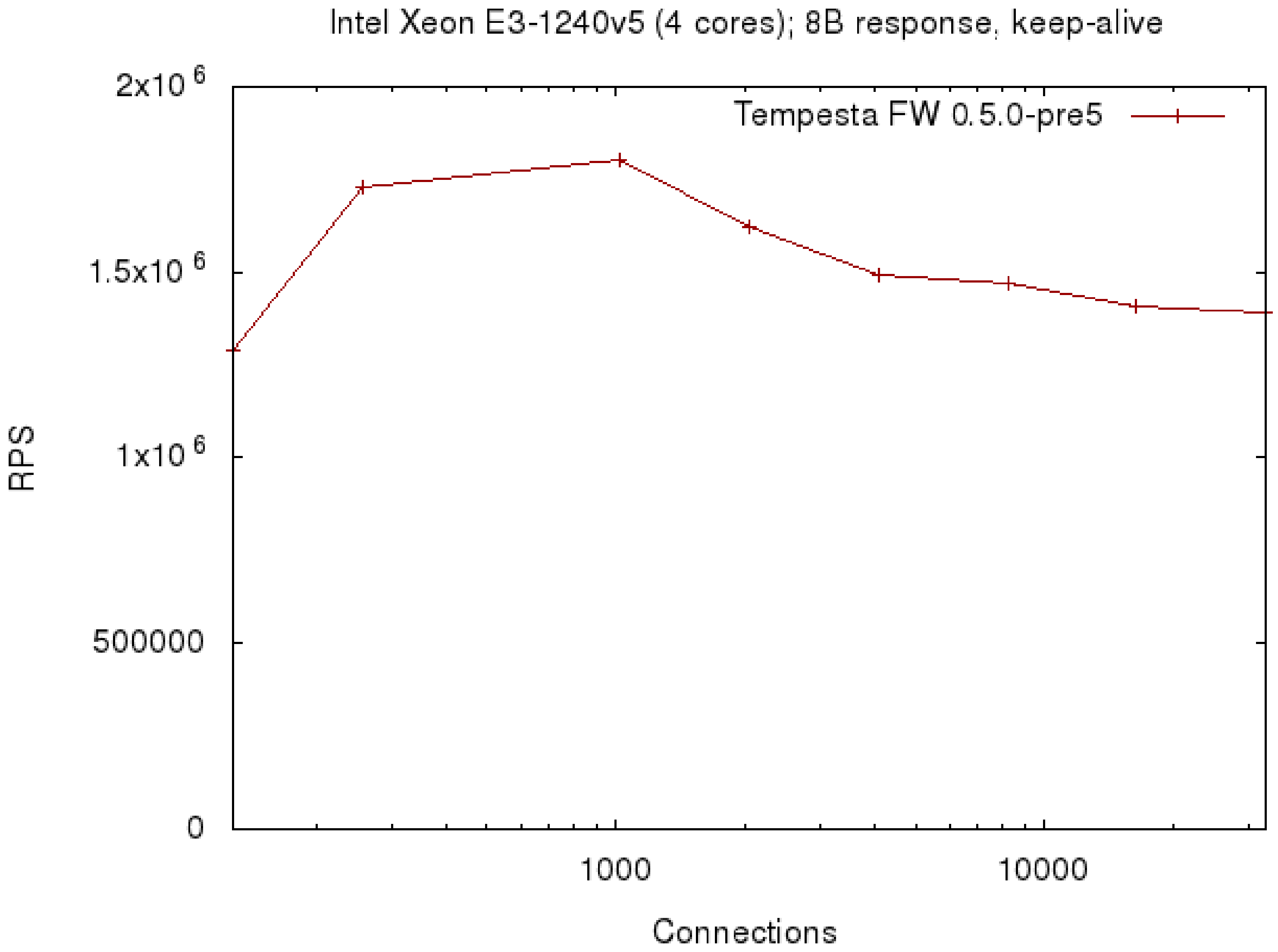
- *https://github.com/tempesta-tech/tempesta/issues/102*

```
if ((req.user_agent =~ /firefox/i
     || req.cookie !~ /^our_tracking_cookie/)
    && (req.x_forwarded_for != "1.1.1.1"
        || client.addr == 1.1.1.1))
        # Block the client at IP layer, so it will be filtered
        # efficiently w/o further HTTP processing.
        tdb.insert("ip_filter", client.addr, evict=10000);
```

- Nftables integration via `mark`
  https://github.com/tempesta-tech/tempesta/issues/760

Tempesta Technologies

Performance

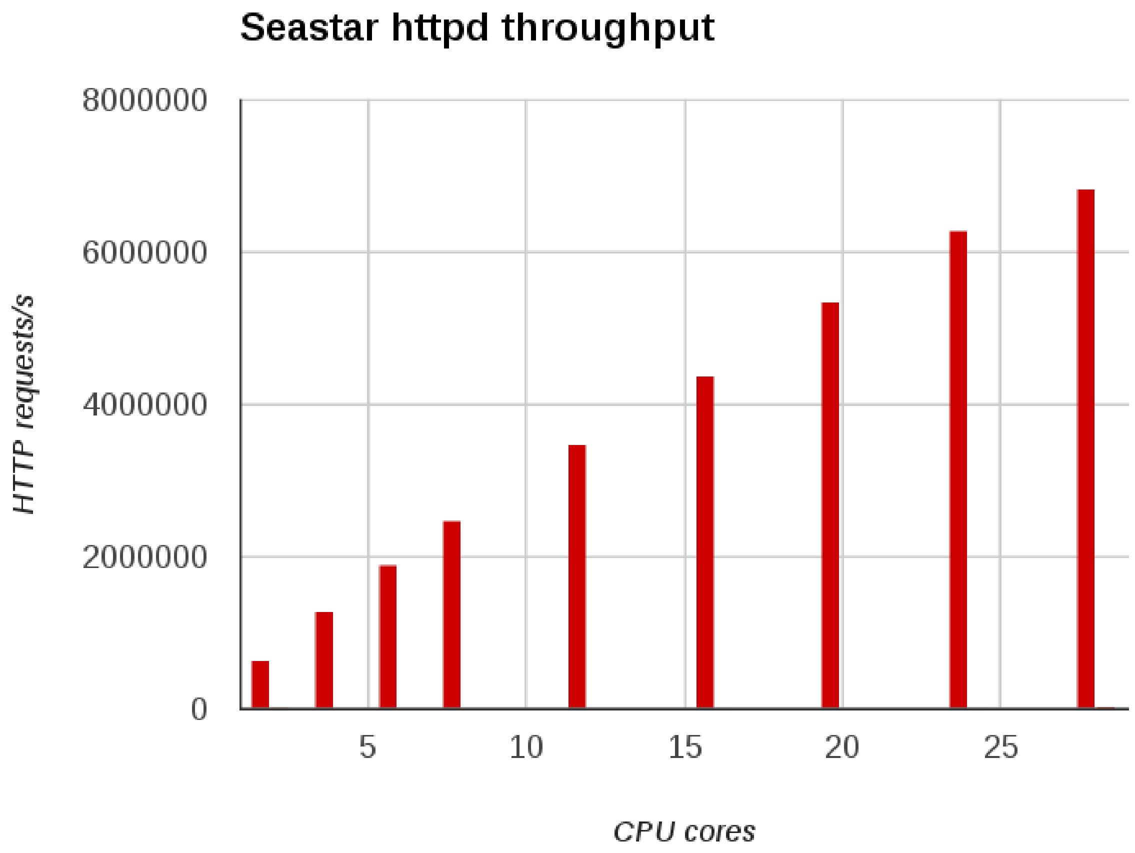


https://github.com/tempesta-tech/tempesta/wiki/HTTP-cache-performance

Tempesta Technologies

Performance analysis

- **~x3 faster than Nginx** (~600K HTTP RPS) for normal Web cache operations
- Must be **much faster to block HTTP DDoS** (*DDoS emulation is an issue*)
- Similar to DPDK/user-space TCP/IP stacks http://www.seastar-project.org/*http-performance/*
- ...bypassing Linux TCP/IP **isn’t the only way** to get a fast Web server
- ...**lives in Linux infrastructure:** LVS, tc, IPtables, eBPF, tcpdump etc.



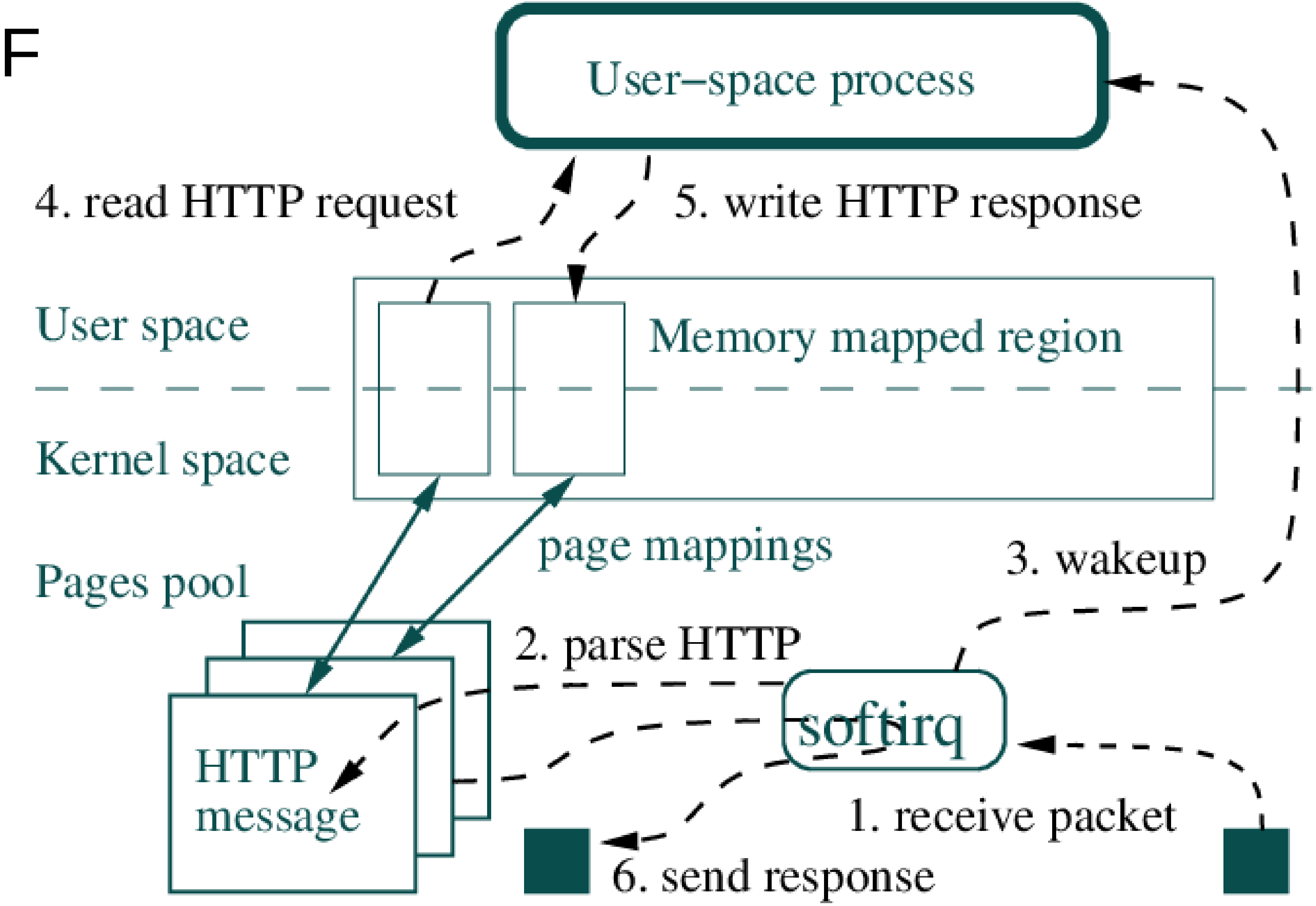
Keep the kernel small

- Just **30K LoC** (compare w/ 120K LoC of BtrFS)
- Only generic and crucial HTTPS logic is in kernel
- Supplementary logic is considered for user space
  - HTTP compression & decompression https://github.com/tempesta-tech/tempesta/issues/636
  - Advanced DDoS mitigation & WAF (e.g. full POST processing)
  - ...other HTTP users (Web frameworks?)
- Zero-copy **kernel-user space transport** for minimizing kernel code

Tempesta Technologies

TODO:
Zero-copy kernel-user space transport

- HTTPS DDoS mitigation & WAF
  - Machine learning clusterization in user space
  - Automatic L3-L7 filtering rules generation



Thanks!

- Web-site: *http://tempesta-tech.com*
- Availability: *https://github.com/tempesta-tech/tempesta*
- Blog: *http://natsys-lab.blogspot.com*
- E-mail: *ak@tempesta-tech.com*

We are hiring!