# Web Acceleration Mechanics

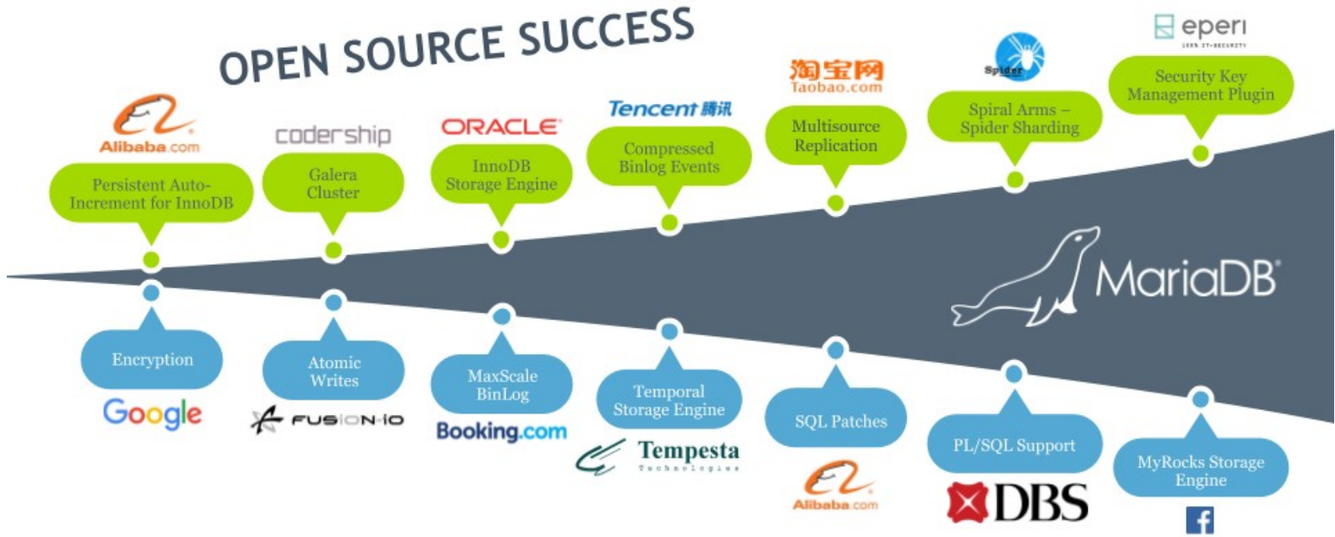
**Alexander Krizhanovsky**

Tempesta Technologies, Inc.

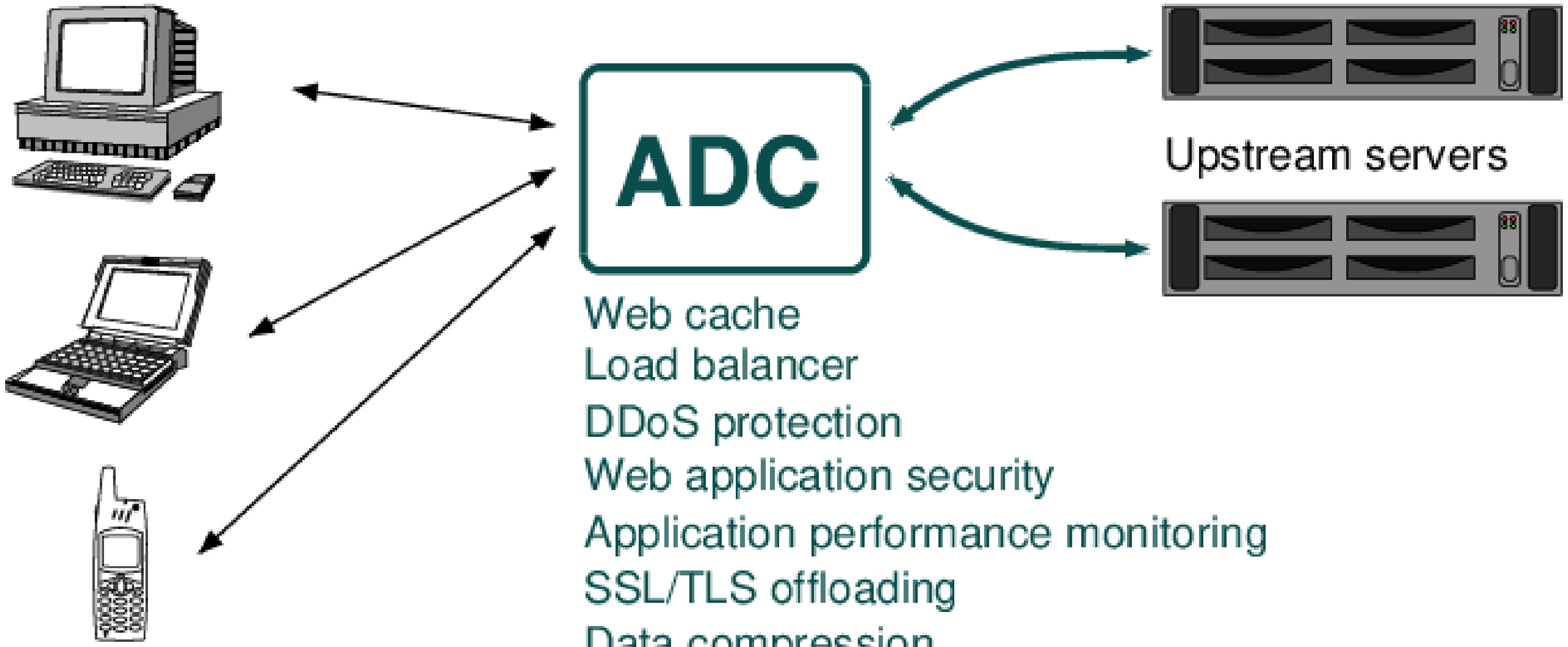
*ak@tempesta-tech.com*

# Who am I?

- CEO at *Tempesta Technologies, INC*
- **Custom software development** since 2008:
  - Network security: WAF, VPN, DPI etc.
    e.g. *Positive Technologies AF*,
    *“Visionar”* **Gartner magic quadrant’15**
  - Databases:
    one of the top **MariaDB** contributors
  - Perfomance tuning
- **Tempesta FW** – Linux *Application Delivery Controller*

# Tempesta FW:
# Application Delivery Controller (ADC)

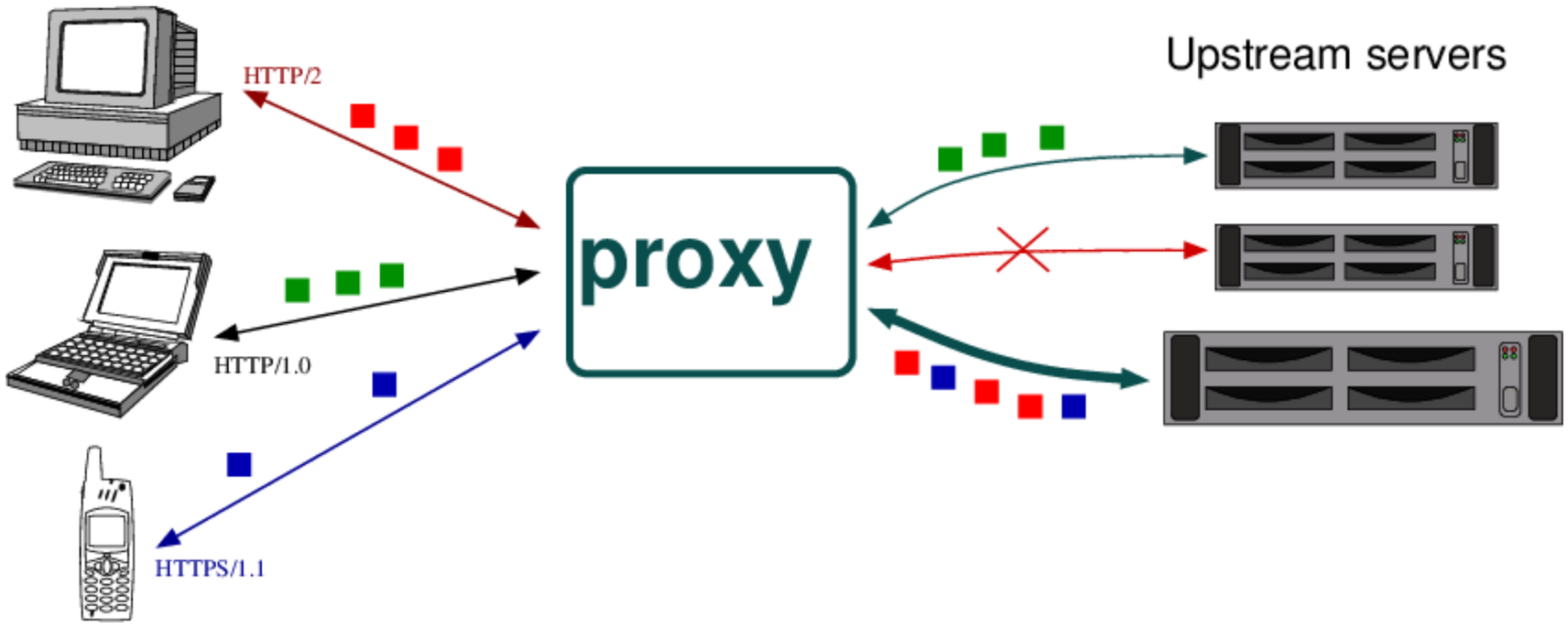



Web cache
Load balancer
DDoS protection
Web application security
Application performance monitoring
SSL/TLS offloading
Data compression
Connections QoS

Tempesta Technologies

# Web accelerator

- Load balancing
- Backend connections management
- TLS termination
- Protocol downgrade
- Cache responses

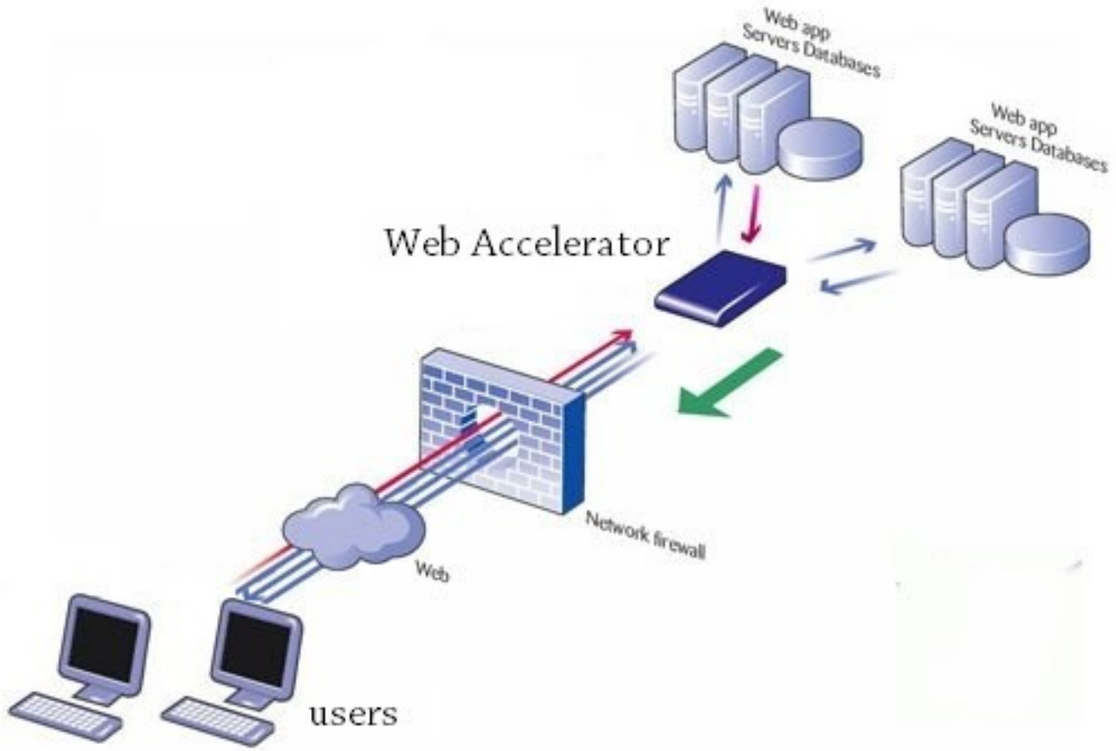





Tempesta Technologies

# Agenda & Disclaimer

- Mechanics:
  - HTTP **connections** & **messages** management
  - HTTP **decoders** & **parsers**
  - Web **caches**
  - **Network I/O**
  - **Multitasking**
  - **TLS**
- The web accelerators are mentioned **only as implementation examples**
- Some software is just more familiar to me

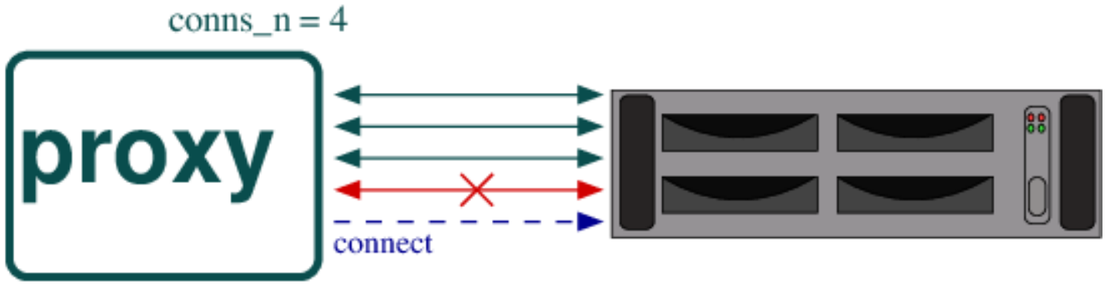



Tempesta Technologies

# HTTP connections & messages management

# Server connections

- New connections or
  **persistent** connections (usual)
- HTTP keep-alive connections
  `Keep-Alive: timeout=5, max=10000`
- Reestablish closed KA connection
- New connections if all are busy
- **N backend connections = N backend requests in-flight**
- DDoS and flash crowds:
  as many server connections as client connections
- Run out of port numbers

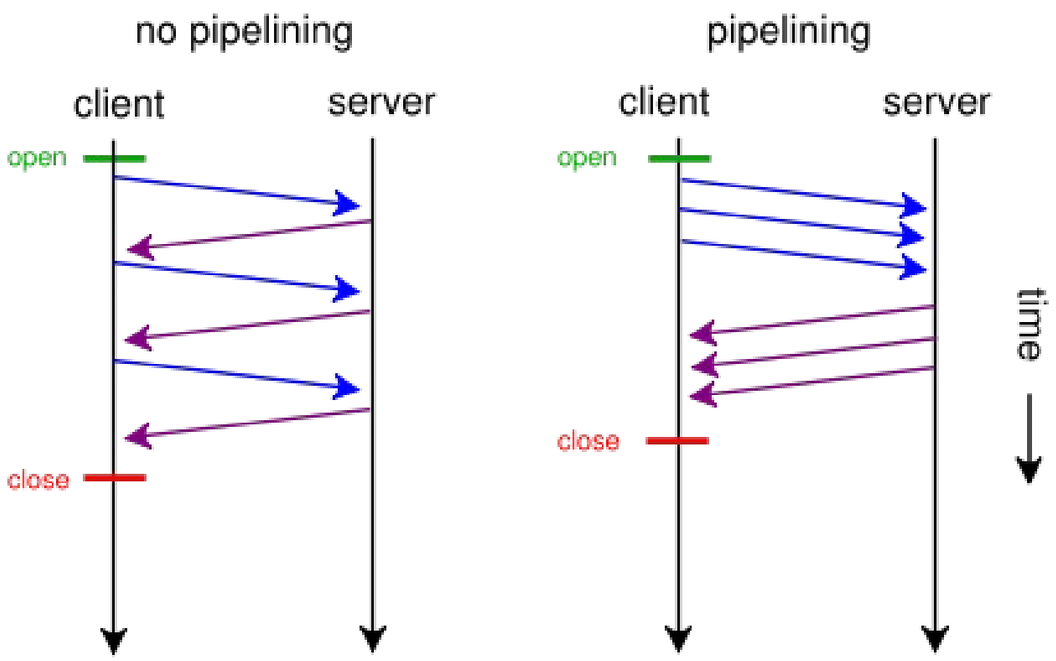





Tempesta Technologies

# HTTP/1 message pipelining

- **Mostly unused by proxies**
- Squid, Tempesta FW, Polipo
- Messages multiplexing
- Forwarding and reforwarding issues
- Security issues
  - Breaking authentication
  - HTTP Response splitting






Tempesta Technologies

# HTTP Response Splitting attack
## (Web cache poisoning)

*[CRLF CRLF]*

```
Client:   /redir_lang.jsp?lang=foobar%0d%0aContent-Length:%200%0d%0a%0d%0a
          HTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0a
          Content-Length:%2019%0d%0a%0d%0a<html>Shazam</html>

Client:   (one more request to inject Shazam into the cache)

Server:   HTTP/1.1 302 Moved Temporarily
          Date: Wed, 24 Dec 2003 15:26:41 GMT
          Location: http://10.1.1.1/by_lang.jsp?lang=foobar
          Content-Length: 0

(server   HTTP/1.1 200 OK
 uses     Content-Type: text/html
 lang     Content-Length: 19
 as is)
          <html>Shazam</html>
```

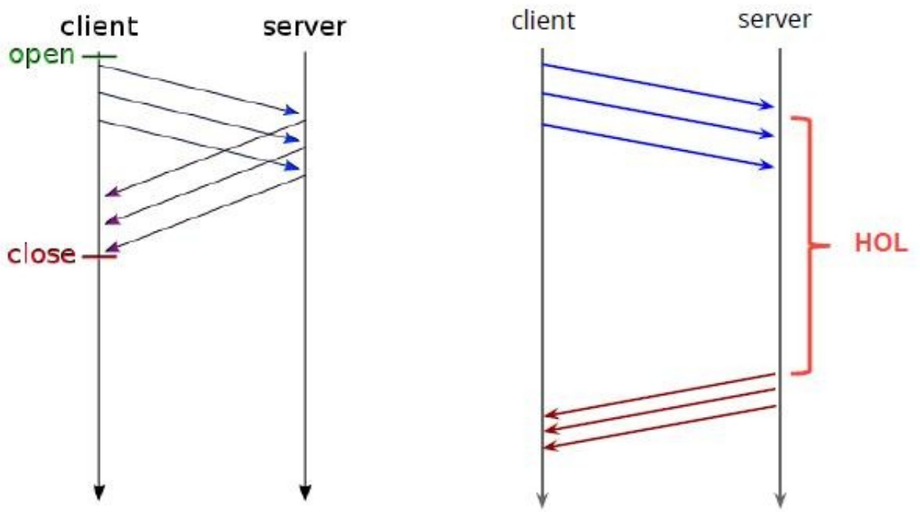
- Decode and validate URI (also for injection attacks)
- **HTTP/2 isn't vulnerable**



Tempesta Technologies

# HTTP/2

- Requests multiplexing instead of pipelining
- **Pros**
  - Responses are sent in *parallel* and in *any order* (no head-of-line blocking)
  - Compression(?)
- **Cons**
  - Less zero copy (different dynamic tables)
- HTTP/2 backend connections (H2O, HAProxy, Envoy)
  - Slow backend connections (e.g. CDN)
  - Slow logic (e.g. dynamic content w/ database access)






Tempesta Technologies

# (Non-)indempotent requests

- RFC 7231 4.2.2: **Non-**idempotent request changes server state
- Idempotent (safe) methods: **GET**, HEAD, TRACE, OPTIONS
  - User responsibility: `GET /action?do=delete`
  - Can be POST (e.g. web-search)
- **Let a user** to specify idempotence of methods and resources:

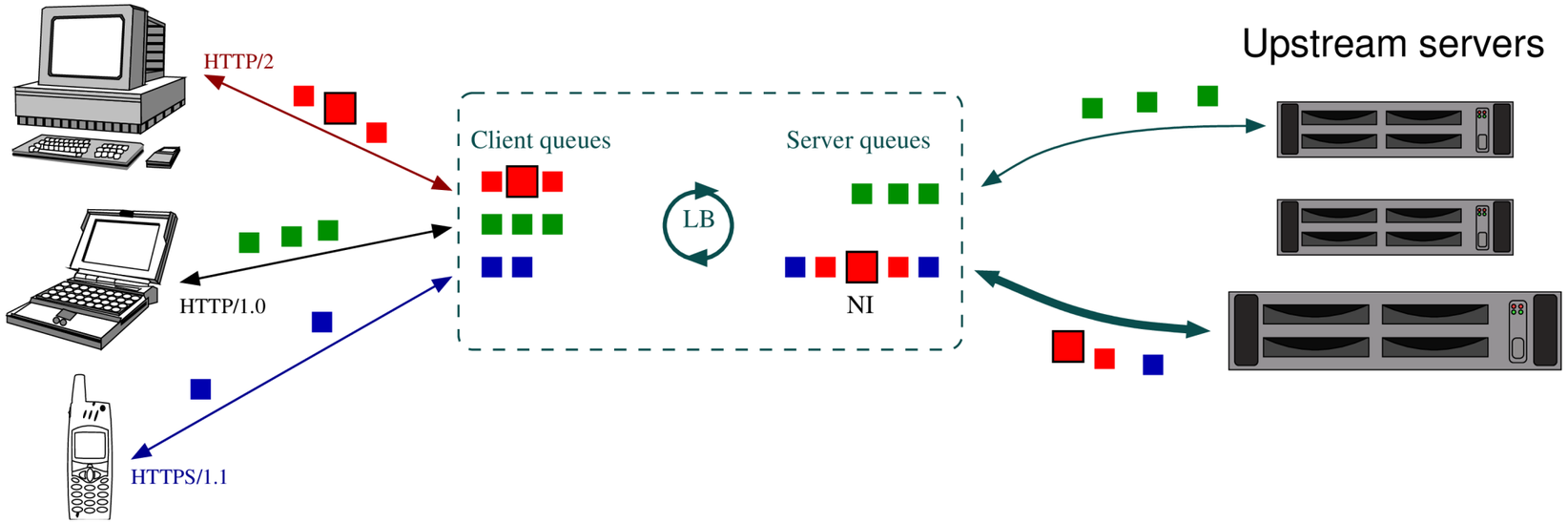
  `nonidempotent GET prefix “/action”`
- Cannot be retried in HTTP/1
- HTTP/2 (RFC 7540 8.1.4): **can retry non-idempotent** requests
  - `GOAWAY` and `RST_STREAM` with `REFUSED_STREAM`

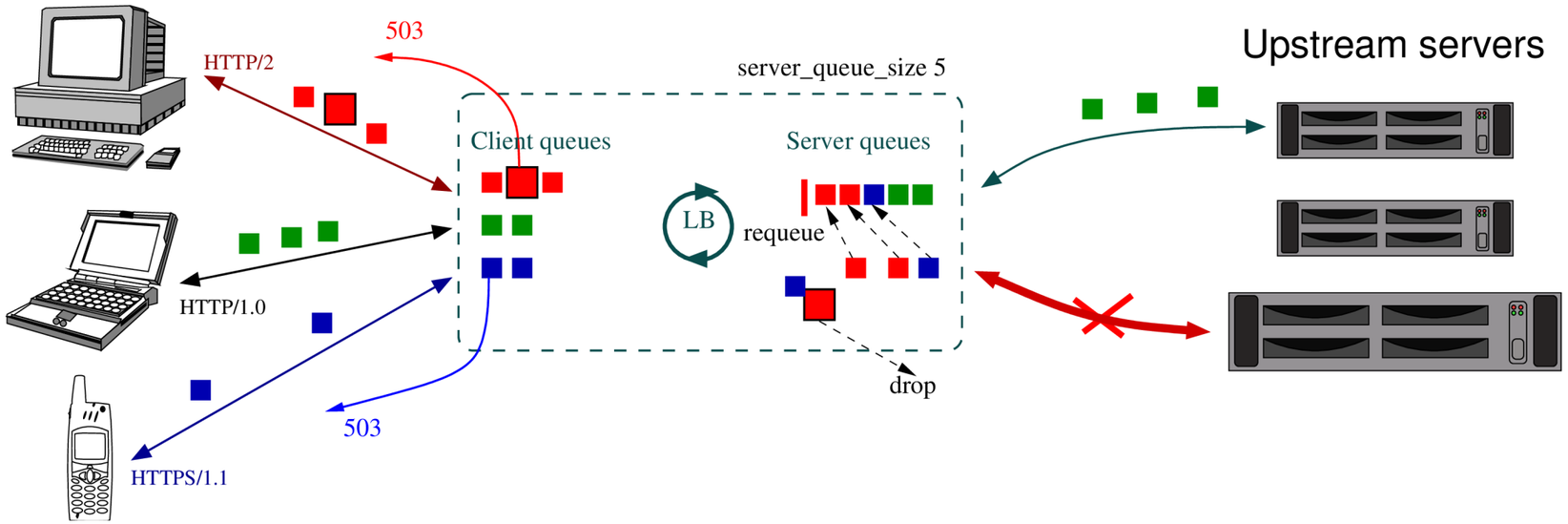
# Resend HTTP requests?

- **Tainted requests**: `server_forward_retries` and `server_forward_timeout`
- **Request-killers** – RFC 7230 6.3.2: *"a client MUST NOT pipeline immediately after connection establishment"*
  - Connection with re-forwarded requests is **non-schedulable**
- **Non-idempotent requests** can be reforwarded

  `server_retry_nonidempotent`
- Error messages keep **order of responses**

# Requests reforwarding
(not for persistent sessions!)



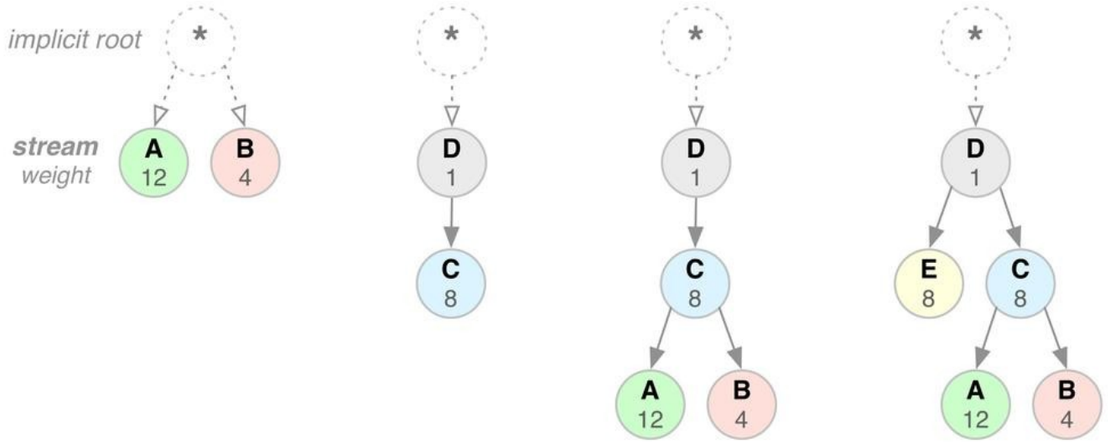



# Proxy response buffering vs streaming

- **Buffering**
  - Seems everyone by default
  - Performance degradation on large messages
  - 200 means OK, no incomplete response
- **Streaming**
  - Varnish, Tengine (patched Nginx)
    `proxy_request_buffering & fastcgi_request_buffering`
  - More performance, but 200 doesn't mean full response

# HTTP/2 prioritization

- **E.g. A=12, B=4**
  - Naive: A(12), A(11), A(10), A(9), A(8), A(7), A(6), A(5), A(4), B(4), A(3), B(3),...
  - WFQ: A(12), A(11), A(10), B(4), A(9), A(8),...
- **Weighted Fair Queue (WFQ)**
  - H2O aproximate in O(1))
  - nghttp2 (and Apache) in O(log(n))



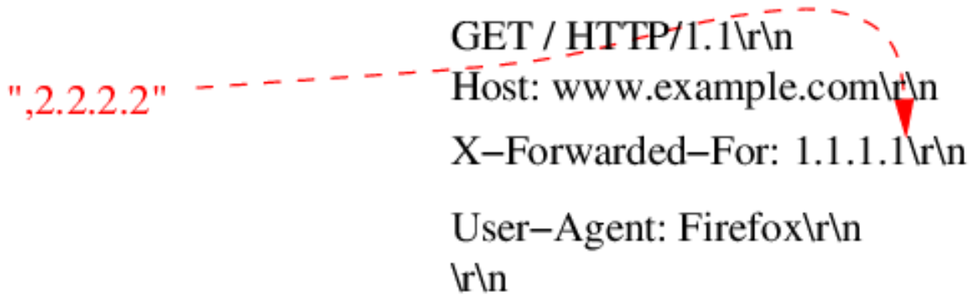

Source: High Performance Browser Networking by Ilya Grigorik



Tempesta Technologies

# HTTP messages adjustment

- Add/remove/update HTTP headers
- Full HTTP message rewriting

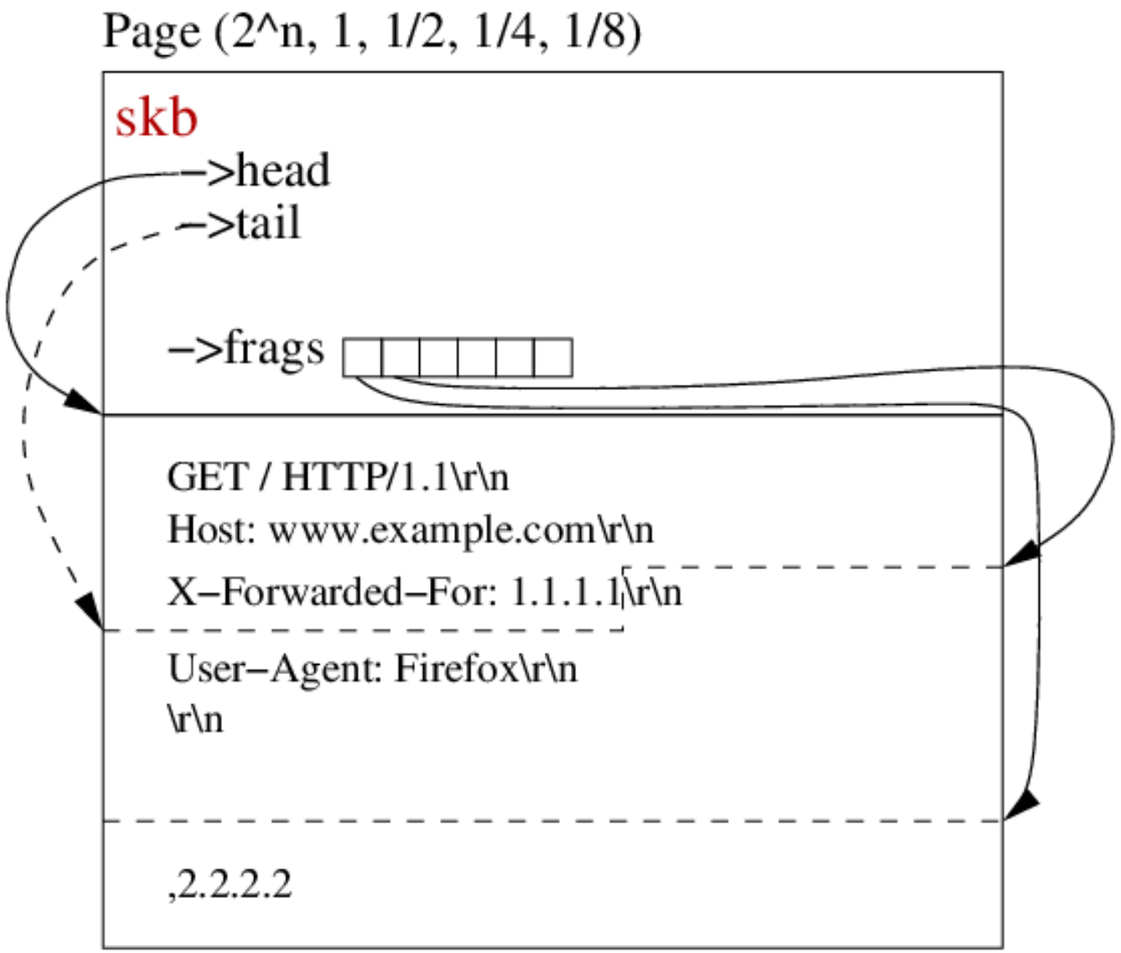





Tempesta Technologies

# HTTP messages adjustment: zero-copy

- Tempesta FW: add/remove/update HTTP headers **without copies**
- Small head and tail memory overheads to avoid dynamic allocations






Tempesta Technologies

# HTTP Parsers

# Multi-pass HTTP parser

```
if (!strncasecmp(hp->hd[HTTP_HDR_URL].b, "http://", 7))
    b = hp->hd[HTTP_HDR_URL].b + 7;
else if (FEATURE(FEATURE_HTTPS_SCHEME) &&
         !strncasecmp(hp->hd[HTTP_HDR_URL].b, "https://", 8))
    b = hp->hd[HTTP_HDR_URL].b + 8;
```

# Switch-driven 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:
        ...
    }
    ...
}
```

# Switch-driven 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:
        ...
    }
    ...
}
```

# Switch-driven 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
}
```

# Switch-driven 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:
        ...
    }
    ...
}
```

# Switch-driven 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
    }
    ...
}
```

# Switch-driven HTTP parser

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

1 2 3 4

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

# HTTP parsers in the wild

- **Case/swith**
  - Nginx, ATS
- **Multi-pass** (glibc calls)
  - Varnish, HAProxy
- **SIMD**
  - Tempesta FW, H2O, CloudFlare
  - Faster on large data (URI, Cookie)
  - Security restrictions against injection attacks (Tempesta FW)
    *SCALE 17x: https://www.slideshare.net/AlexanderKrizhanovsky1/fast-http-string-processing-algorithms*



Tempesta Technologies

# Why HTTP strings matter?

- Usual URI – just a hotel query

```
https://www.booking.com/searchresults.en-us.html?
aid=304142&label=gen173nr-
1FCAEoggI46AdIM1gEaIkCiAEBmAExuAEZyAEP2AEB6AEB-
AECiAIBqAIDuAKAg4DkBcACAQ&sid=686a0975e8124342396dbc1b331
fab24&tmpl=searchresults&ac_click_type=b&ac_position=0&ch
eckin_month=3&checkin_monthday=7&checkin_year=2019&checko
ut_month=3&checkout_monthday=10&checkout_year=2019&class_
interval=1&dest_id=20015107&dest_type=city&dtdisc=0&from_
sf=1&group_adults=1&group_children=0&inac=0&index_postcar
d=0&label_click=undef&no_rooms=1&postcard=0&raw_dest_type
=city&room1=A&sb_price_type=total&sb_travel_purpose=busin
ess&search_selected=1&shw_aparth=1&slp_r_match=0&src=inde
x&srpvid=e0267a2be8ef0020&ss=Pasadena%2C%20California%2C
%20USA&ss_all=0&ss_raw=pasadena&ssb=empty&sshis=0&nflt=hot
elfacility%3D107%3Bmealplan%3D1%3Bpri%3D4%3Bpri
%3D3%3Bclass%3D4%3Bclass%3D5%3Bpopular_activities
%3D55%3Bhr_24%3D8%3Btdb%3D3%3Breview_score
%3D70%3Broomfacility%3D75%3B&rsf=
```

- How about tons of such queries? (DDoS)
- How about injections?

```
/redir_lang.jsp?lang=foobar%0d%0aContent-Length:%200%0d
%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/
html%0d%0aContent-Length:%2019%0d%0a%0d%0a<html>Shazam</
html>
```

```
case sw_check_uri:
    if (usual[ch >> 5] & (1U << (ch & 0x1f)))
        break;
    switch (ch) {
    case '/':
        r->uri_ext = NULL;
        state = sw_after_slash_in_uri;
        break;
    case '.':
        r->uri_ext = p + 1;
        break;
    case ' ':
        r->uri_end = p;
        state = sw_check_uri_http_09;
        break;
    case CR:
        r->uri_end = p;
        r->http_minor = 9;
        state = sw_almost_done;
        break;
    case LF:
        r->uri_end = p;
        r->http_minor = 9;
        goto done;
    case '%':
        r->quoted_uri = 1;
    ...
```



Tempesta Technologies

# Let’s check

- Reasonable HTTP request

```
./wrk -t 4 -c 128 -d 60s --header 'Connection: keep-alive' --header 'Upgrade-Insecure-Requests: 1'
--header 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.116 Safari/537.36' --header 'Accept: text/html,application/xhtml+xml,
application/xml;q=0.9,image/webp,*/*;q=0.8' --header 'Accept-Encoding: gzip, deflate, sdch'
--header 'Accept-Language: en-US,en;q=0.8,ru;q=0.6' --header 'Cookie: a=sdfasd; sdf=3242u389erfhhs;
djcnjhe=sdfsdafsdjfb324te1267dd' 'http://192.168.100.4:9090/searchresults.en-us.html?
aid=304142&label=gen173nr-1FCAEoggI46AdIM1gEaIkCiAEBmAExuAEZyAEP2AEB6AEB-AECiAIBqAIDuAKAg4DkBcACAQ
&sid=686a0975e8124342396dbc1b331fab24&tmpl=searchresults&ac_click_type=b&ac_position=0&checkin_month=3&che
ckin_monthday=7&checkin_year=2019&checkout_month=3&checkout_monthday=10&checkout_year=2019&class_interval=
1&dest_id=20015107&dest_type=city&dtdisc=0&from_sf=1&group_adults=1&group_children=0&inac=0&index_postcard
=0&label_click=undef&no_rooms=1&postcard=0&raw_dest_type=city&room1=A&sb_price_type=total&sb_travel_purpos
e=business&search_selected=1&shw_aparth=1&slp_r_match=0&src=index&srpvid=e0267a2be8ef0020&ss=Pasadena%2C
%20California%2C%20USA&ss_all=0&ss_raw=pasadena&ssb=empty&sshis=0&nflt=hotelfacility%3D107%3Bmealplan
%3D1%3Bpri%3D4%3Bpri%3D3%3Bclass%3D4%3Bclass%3D5%3Bpopular_activities%3D55%3Bhr_24%3D8%3Btdb
%3D3%3Breview_score%3D70%3Broomfacility%3D75%3B&rsf='
```

- Even for simple HTTP parser

```
8.62%  nginx        [.] ngx_http_parse_request_line
2.52%  nginx        [.] ngx_http_parse_header_line
1.42%  nginx        [.] ngx_palloc
0.90%  [kernel]     [k] copy_user_enhanced_fast_string
0.85%  nginx        [.] ngx_strstrn
0.78%  libc-2.24.so [.] _int_malloc
0.69%  nginx        [.] ngx_hash_find
0.66%  [kernel]     [k] tcp_recvmsg
```



Tempesta Technologies

# HTTP/{2,3} Decoders

# HPACK (HTTP/2) & QPACK (QUIC)

- Huffman encoding – skewed bytes => no SIMD
- Dynamic table: at least 4KB overhead for each connection
  - HPACK bomb (**2016**): **N** requests with **M** dynamic table indexed headers of size **S** => OOM with **N** * **M** * **S** bytes
- Makes sense for requests only (~1.4% improvement for responses) *https://blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2/*
- Still have to process strings, even for static headers :(

# HTTP/2 as the first class citizen

- HTTP/2 as an add-on to HTTP/1

```
// Cookie is an indexed header in the static table
static ngx_str_t cookie = ngx_string("cookie");
if (ngx_memcmp(header->name.data, cookie.data, cookie.len) == 0)
```

- Fast headers lookup using static table indexes (H2O, Tempesta FW)
- Store web cache entries in HTTP/2-friendly format (Tempesta FW)



Tempesta Technologies

# Web Caching

# To cache

- Determined by `Cache-Control` and `Pragma`
- static (e.g. video, images, CSS, HTML)
- *some* dynamic
- Negative results (e.g. 404)
- Permanent redirects
- Incomplete results (206, RFC 7233 Range Requests)
- Methods: **GET**, POST, whatever
- `GET /script?action=delete` – this is your responsibility
  *(but some servers don't cache URIs w/ arguments)*

# Not to cache

- Determined by `Cache-Control` and `Pragma`
- Explicit **no-cache** or **private** directives
- Responses to Authenticated requests
- Unsafe methods (RFC 7231 4.2.1)
  (safe methods: GET, HEAD, OPTIONS, TRACE)
- Set-Cookie (?)

# Cache Set-Cookie?

- **Varnish**, **Nginx**, **ATS don't cache** responses w/ Set-Cookie by default
- **Tempesta FW**, **mod_cache**, **Squid do cache** such responses
- **RFC 7234**, 8 Security Considerations:
  Note that the Set-Cookie response header field [RFC6265] **does not inhibit caching;** a cacheable response with a Set-Cookie header field can be (and often is) used to satisfy subsequent requests to caches. Servers who wish to control caching of these responses are encouraged to **emit appropriate Cache-Control** response header fields.



Tempesta Technologies

# Cache POST?

- Discussion: *https://www.mnot.net/blog/2012/09/24/caching_POST*
  - RFC 7234 4.4: URI must be invalidated
- Original eBay case: search API with **too many parameters for GET** *https://tech.ebayinc.com/engineering/caching-http-post-requests-and-responses/*
- **Idempotent** POST (e.g. web-serarch) – just like GET
- **Non-idempotent** POST (e.g. blog comment) – cache response for following GET

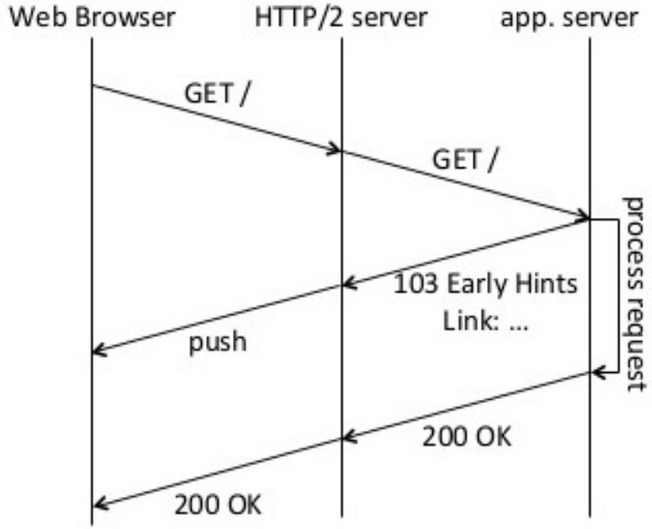
# Cache entries freshness

- **RFC 7234:** `freshness_lifetime > current_age`
- **Freshness calculation** headers: `Last-Modified`, `Date`, `Age`, `Expires`
- **Revalidation:**
  - Conditional requests (RFC 7232, e.g. **If-Modified-Since**)
  - Background activity or on-request job
    Nginx: `proxy_cache_background_update`
- Sometimes it's OK to return **stale cache entries:**
  Nginx: `proxy_cache_use_stale`

# HTTP/2 server PUSH

- **HTTP/2 client ↔ proxy ↔ HTTP/1.1 server** (e.g. H2O, Nginx)
  - Try with `PUSH_PROMISE` and stop by `RST_STREAM`
  - Apache HTTPD: “diary” what has been pushed (`H2PushDiarySize`)
  - H2O CASPer tracks client cache state via cookies
- **Link** header (RFC 5988)
  - `Preload` (draft) https://w3c.github.io/preload/
- **103 Early Hints** (RFC 8297, H2O)

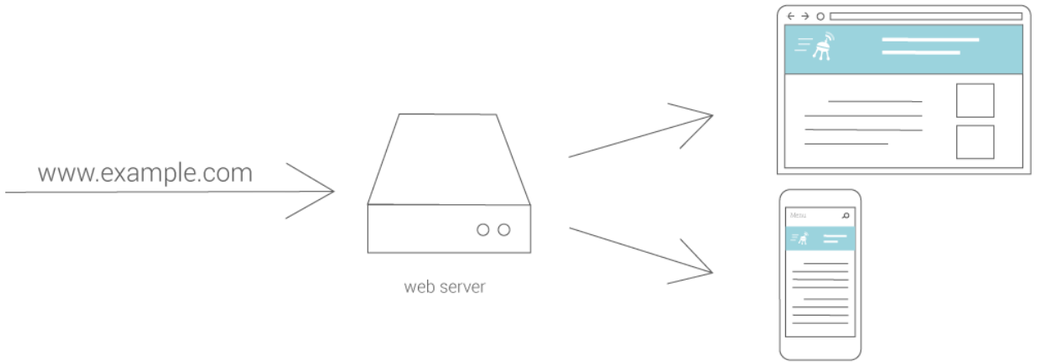
```
GET / HTTP/1.1
Host: example.com

HTTP/1.1 103 Early Hints
Link: </style.css>; rel=preload

HTTP/1.1 200 OK
Content-Type: text/html
Link: </style.css>; rel=preload

<!DOCTYPE HTML>
...
```




Source: “Reorganizing Website Architecture for HTTP/2 and Beyond” Kazuho Oku

# Vary

- *Accept-Language* – return localized version of page (no need /**en**/index.html)
- *User-Agent* – mobile vs desktop (bad!)
- *Accept-Encoding* – don't send compressed page if browser doesn't understaind it
- **Seconday keys**: say “hello” to databases
- Request headers normalization is required!






Tempesta Technologies

# Cache storage

- **Plain files** (Nginx, Squid, Apache HTTPD)

```
/cache/0/1d/4af4c50ff6457b8cabfdcd32d0b2f1d0
/cache/5/2e/9f351cdfc8027852656aac5d3f9372e5
/cache/f/22/554a5c654f189c1630e49834c25ae229
```

  - Meta-data in RAM
  - Filesystem database
  - Easy to manage
- **Virtual memory** (Varnish): `mmap(2)`, `malloc(3)` – no persistency
- **Database** (Apache Traffic Server, Tempesta FW)
  - Faster access
  - Persistency (no ACID guarantees)

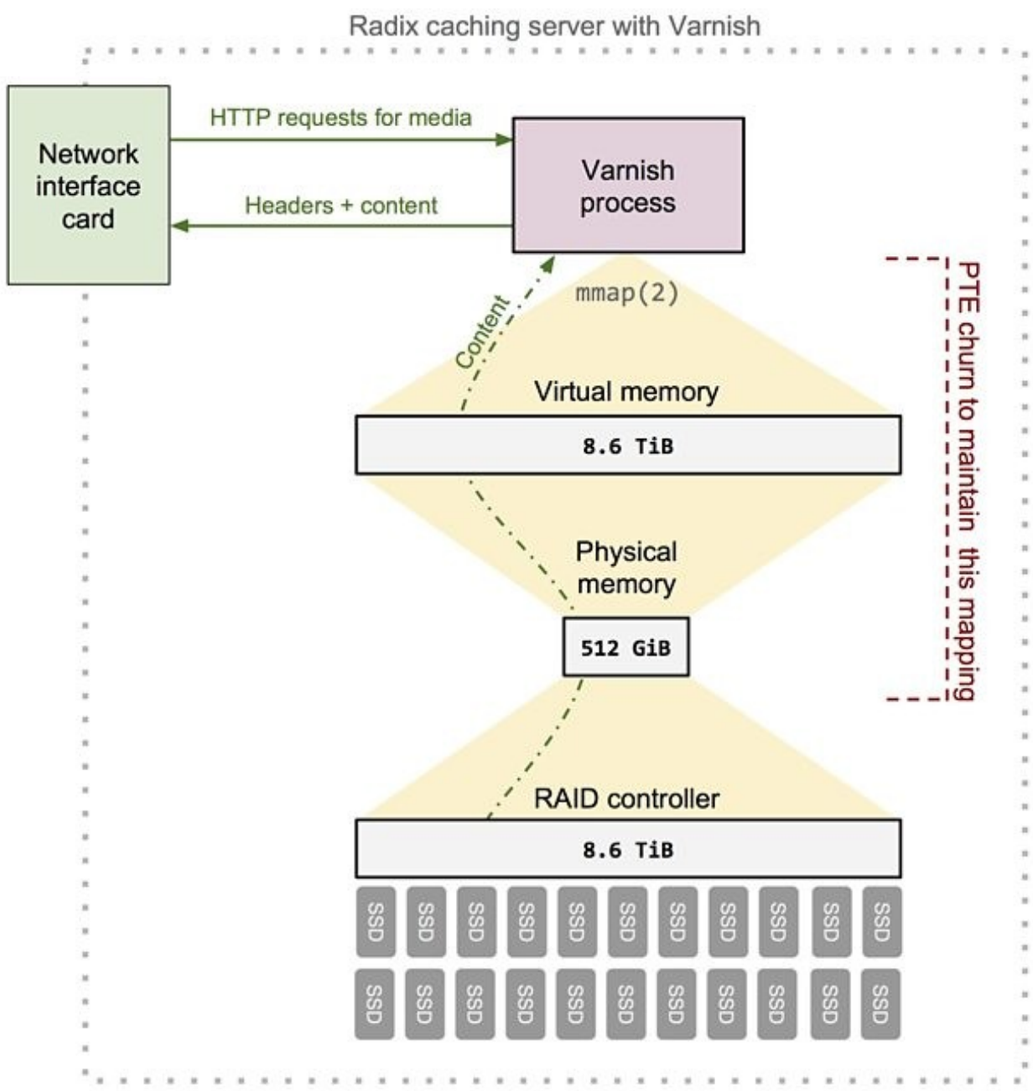


Tempesta Technologies

# Cache storage: mmap(2)

- *http://www.bbc.co.uk/blogs/internet/entries/17d22fb8-cea2-49d5-be14-86e7a1dcde04*
- 48 CPUs, 512GB RAM, 8TB SSD
- Basically the same issues with any `mmap()`-based database

  *https://www.percona.com/live/data-performance-conference-2016/sessions/linux-kernel-extension-databases*



# Requests coalescing

- Varnish, Nginx
- Cases:
  - Cold web cache
  - Large respones (e.g. video streaming)
- Reduces thundering herd of requests to an upstream

# Network I/O
# &
# Multitasking

# IO & multitasking

- Updated *“Choosing A Proxy Server”*, ApacheCon 2014, Bryan Call

| | ATS | Nginx | Squid | Varnish | Apache HTTPD | Tempesta |
|---|---|---|---|---|---|---|
| Threads | X | * | | per-session! | X | |
| Events | X | X | X | partial | X | data-driven |
| Processes | | X | X | | X | |
| Softirq | | | | | | X |

* Nginx 1.7.11 introduced thread pool, but for asynchronous I/O only



Tempesta Technologies

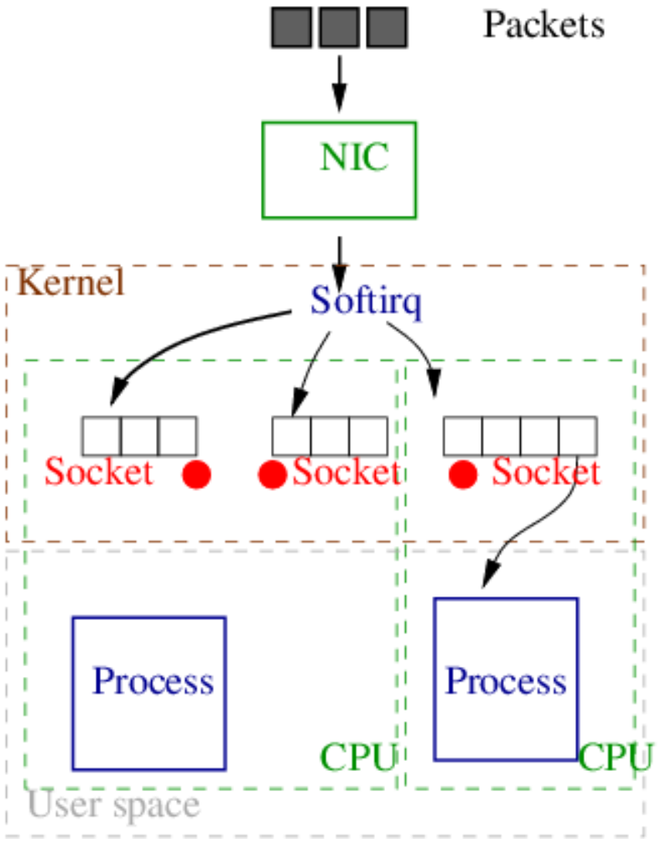
# Threads vs Processes vs Softirq

- Thread per session (e.g. Varnish)
  - Threads overhead: ~6 pages, log(n) scheduler
  - **No** slow request **starvation** (e.g. asynchronous DDoS)
- Meltdown (reading kernel memory from user space): `CONFIG_PAGE_TABLE_ISOLATION` (KPTI, **default** in modern kernels)
  - no lazy TLB as previously, PCID instead
  - ~**40%** perf degradation (*https://mariadb.org/myisam-table-scan-performance-kpti/*)
  - Tempesta FW: no degradation for the in-kernel workload
  - Do you need the protection for a single-user edge server?



Tempesta Technologies

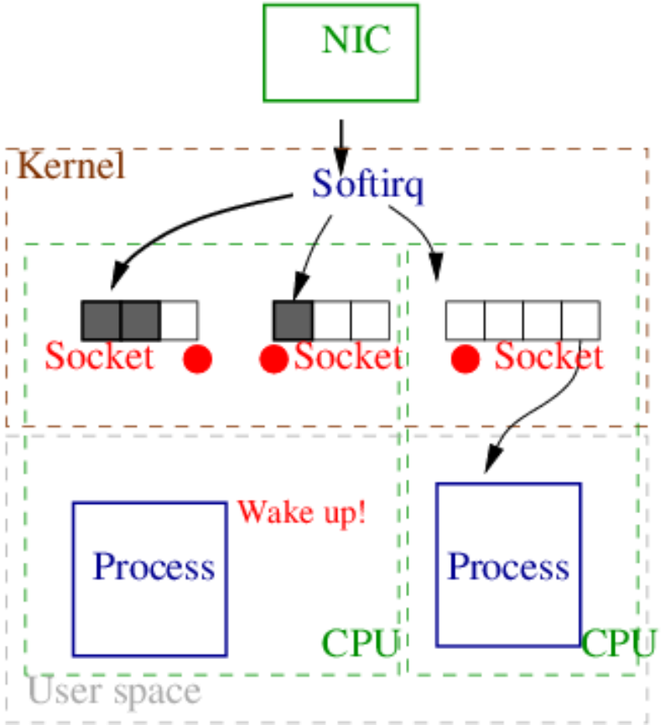
# Network asynchronous I/O






Tempesta Technologies

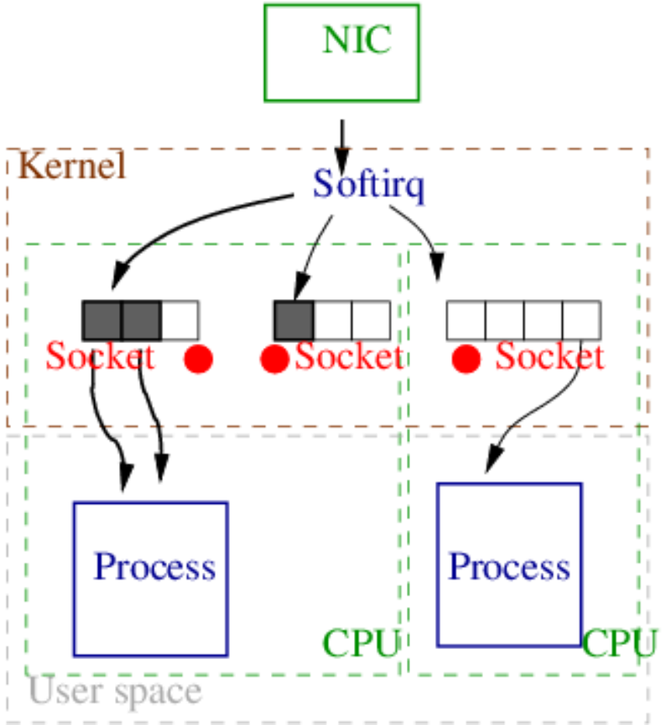
# Network asynchronous I/O






Tempesta Technologies

# Network asynchronous I/O

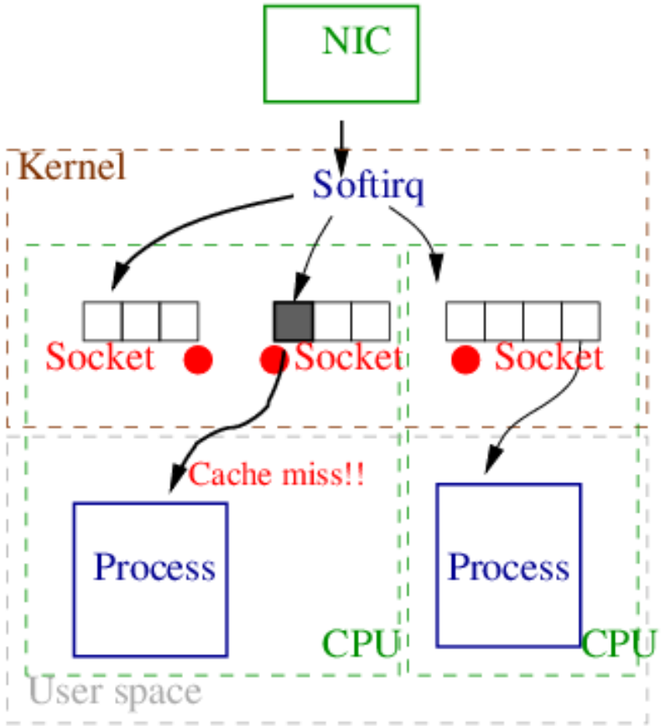





Tempesta Technologies

# Network asynchronous I/O

- Web cache also resides in CPU caches and evicts requests

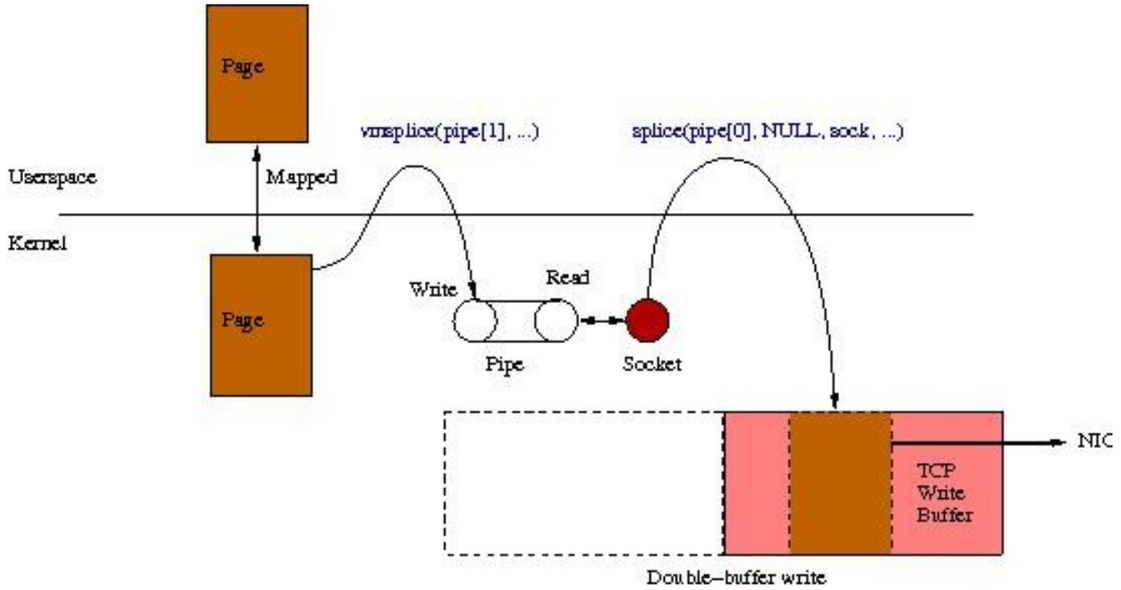





Tempesta Technologies

# vmsplice(2) zero-copy transmission

- 2 system calls instead of 1 (KPTI!)
- Double buffering
- For large transfers only
- Examples: HAProxy




```
vmsplice(pipe_[1], iov, num, 0);
for (int r; data_len; data_len -= r)
  r = splice(pipe_[0], NULL, sd, NULL, data_len, 0);
```

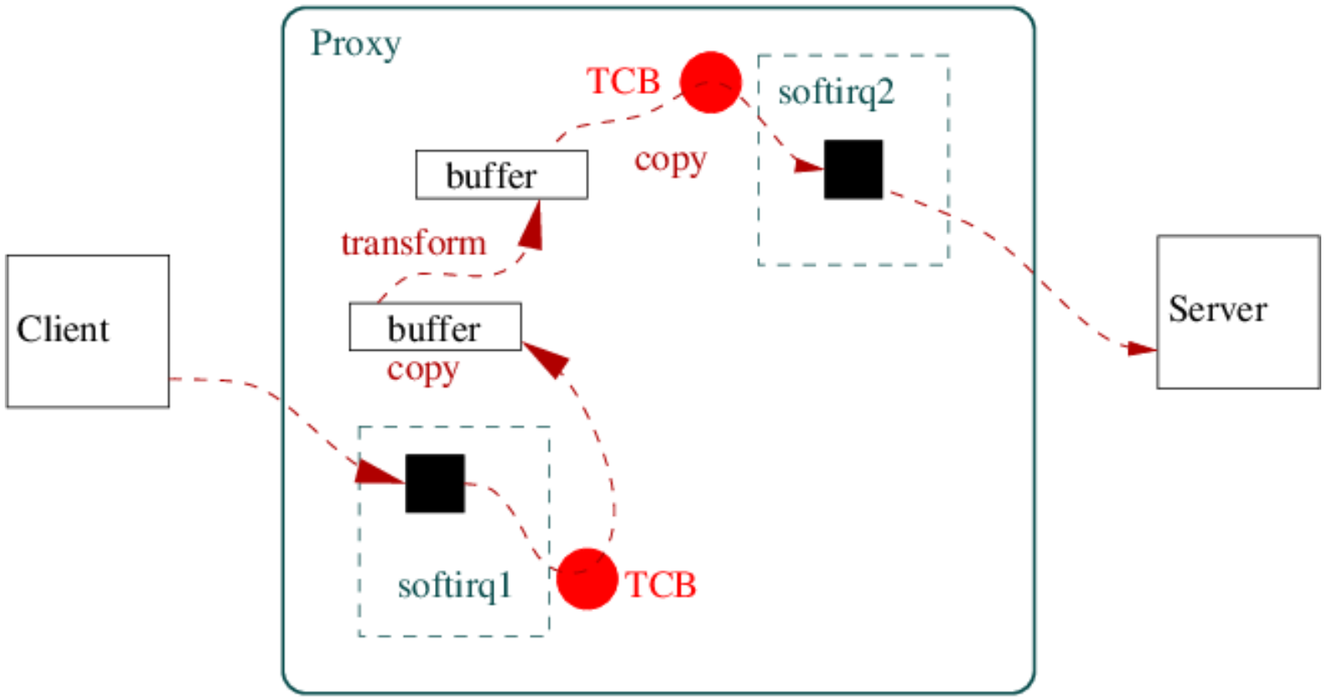
```
# ./nettest/xmit -s65536 -p1000000  127.0.0.1 5500
xmit: msg=64kb, packets=1000000 vmsplice() -> splice()
usr=9259, sys=6864, real=27973

# ./nettest/xmit -s65536 -p1000000 -n 127.0.0.1 5500
xmit: msg=64kb, packets=1000000 send()
usr=8762, sys=25497, real=34261
```



Tempesta Technologies

# User space HTTP proxying

1. Receive request at CPU1
2. Copy request to user space
3. Update headers
4. Copy request to kernel space (except HAProxy & Tempesta FW)
5. Send the request from CPU2



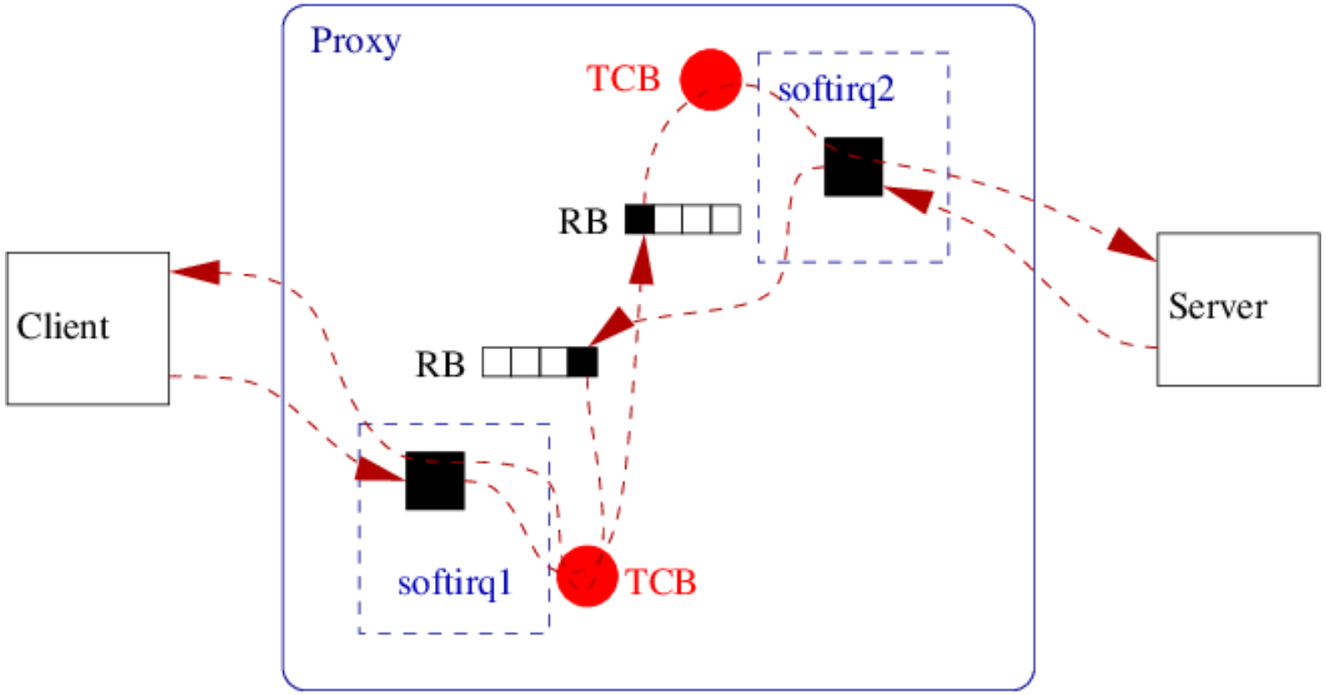

- **>= 3 data copies**
- Access TCP control blocks and data buffers from **different CPUs**



Tempesta Technologies

# Tempesta FW: zero-copy proxying

- 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

# Logging

- ATS, Nginx, Squid, Apache HTTPD: *write(2)*
- Varnish: logs in shared memory → varnishlog
- Tempesta FW: dmesg (in-memory in further releases)

# TLS

# TLS termination

- Native TLS termination: Nginx, HAProxy, H2O, ATS,...
- Varnish Hitch (no TLS in Varnish cache by design)
- Tempesta TLS – kernel TLS termination *https://netdevconf.info/0x14/session.html?talk-performance-study-of-kernel-TLS-handshakes*
- Intel QuickAssist Technology (QAT) – crypto & compression acceleration
  - PCIe adapters, 89xx chipset, Xeon N
  - OpenSSL & zLib & Nginx patches + user-space library



Tempesta Technologies

# SSL/TLS: (zero-)copying

- User-kernel space copying
  - Copy network data to user space
  - Encrypt/decrypt it
  - Copy the data to kernel for transmission
- **Kernel-mode TLS** (Linux kTLS)
  *Facebook*: *https://lwn.net/Articles/666509/*
  - Eliminates ingress & egress data copyings
  - **Nobody** in user space uses it
    *(OpenSSL patches are in progress!)*
  - Unaware about TCP transmission state (cwnd & rwnd)

# TLS record size & TCP

- TLS record is 16KB by default => several TCP segments
- TLS decrypts only full records
- TCP congestion & receive windows can cause last segment delays

```
▽ [8 Reassembled TCP Segments (11221 bytes): #169(1460), #170(1460), #172(1460), #174(1460),
                                             #175(1460), #177(1460), #179(1460), #180(1001)]
    [Frame: 169, payload: 0-1459 (1460 bytes)]
    [Frame: 170, payload: 1460-2919 (1460 bytes)]
    [Frame: 172, payload: 2920-4379 (1460 bytes)]
    [Frame: 174, payload: 4380-5839 (1460 bytes)]
    [Frame: 175, payload: 5840-7299 (1460 bytes)]
    [Frame: 177, payload: 7300-8759 (1460 bytes)]
    [Frame: 179, payload: 8760-10219 (1460 bytes)]
    [Frame: 180, payload: 10220-11220 (1001 bytes)]
    [Segment count: 8]
    [Reassembled TCP length: 11221]
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 11216
      Encrypted Application Data: 07ed92e420530da2e2755a5b5372ef32b53e0d4e7c20c3d8...
```

Source: https://hpbn.co/transport-layer-security-tls/



Tempesta Technologies

# TCP & TLS dynamic record size

- TCP CWND & RWND must be used to avoid multiple RTTs
- **Typical dynamic TLS record size strategies**
  - Static size (Nginx)
  - Dynamic (HAProxy, ATS, H20): static algorithms (no cwnd)
  - kTLS: depends on available wmem for the socket
- QUIC: TLS record = QUIC packet (no need for TLS dynamic records)
- **Tempesta FW** (pros of being in-TCP/IP stack)
  - TCP send queue, NET_TX_SOFTIRQ
  - min(sndbuff, cwnd, rwnd)
  - http://www.netdevconf.org/2.1/papers/https_tcpip_stack.pdf

# References


- **Kernel HTTP/TCP/IP stack for HTTP DDoS mitigation**, Alexander Krizhanovsky, Netdev 2.1, *https://netdevconf.info/2.1/session.html?krizhanovsky*
- **HTTP requests proxying**, Alexander Krizhanovsky, *https://natsys-lab.blogspot.com/2018/03/http-requests-proxying.html*
- **HTTP Strings Processing Using C, SSE4.2 and AVX2**, Alexander Krizhanovsky, *https://natsys-lab.blogspot.com/2016/10/http-strings-processing-using-c-sse42.html*
- **Fast Finite State Machine for HTTP Parsing**, Alexander Krizhanovsky, *https://natsys-lab.blogspot.com/2014/11/the-fast-finite-state-machine-for-http.html*
- **Reorganizing Website Architecture for HTTP/2 and Beyond**, Kazuho Oku, *https://www.slideshare.net/kazuho/reorganizing-website-architecture-for-http2-and-beyond*
- **Server Implementationsof HTTP/2 Priority**, Kazuhiko Yamamoto, *https://www.mew.org/~kazu/material/2015-http2-priority2.pdf*
- **NGINX structural enhancements for HTTP/2 performance**, CloudFlare, *https://blog.cloudflare.com/nginx-structural-enhancements-for-http-2-performance/*

# Thanks!

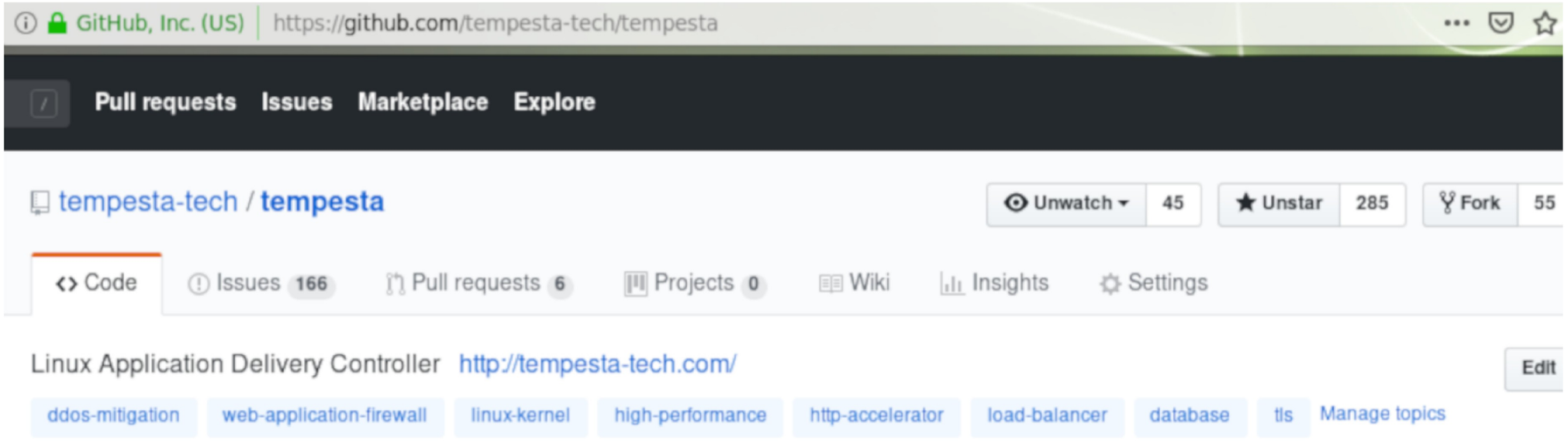
*ak@tempesta-tech.com*

*http://tempesta-tech.com*

*alexander-krizhanovsky*

*a_krizhanovsky*

**Custom software development:** ***tempesta-tech.com/c++-services***






Tempesta Technologies