Fast HTTP strings

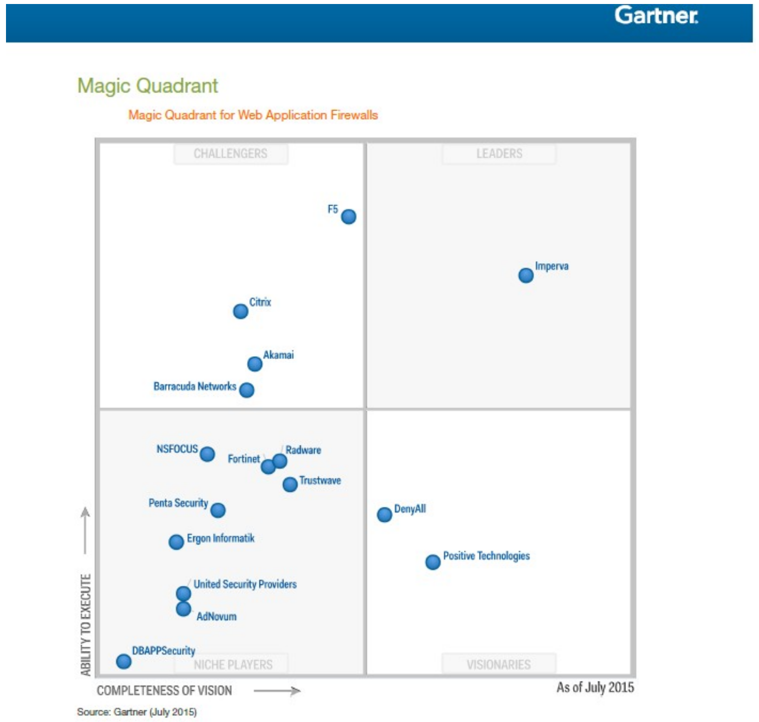
Alexander Krizhanovsky

Tempesta Technologies, Inc.

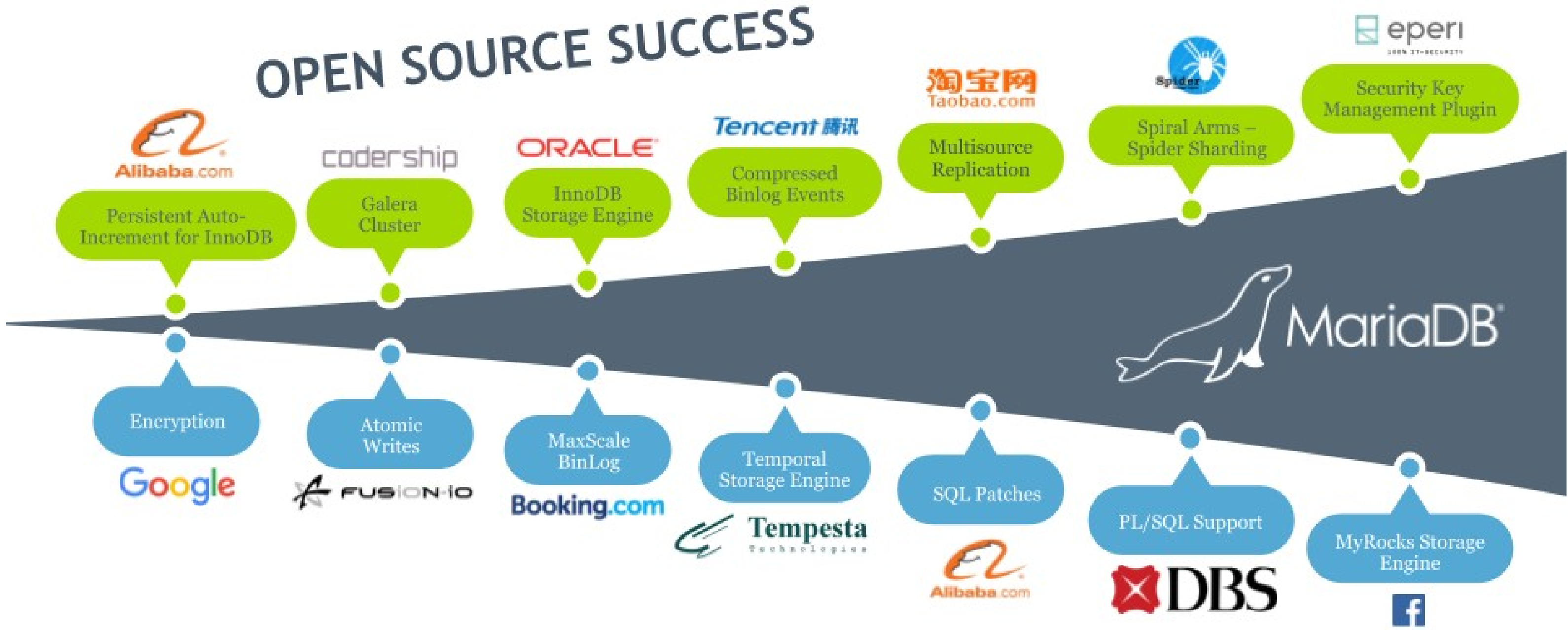
ak@tempesta-tech.com

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

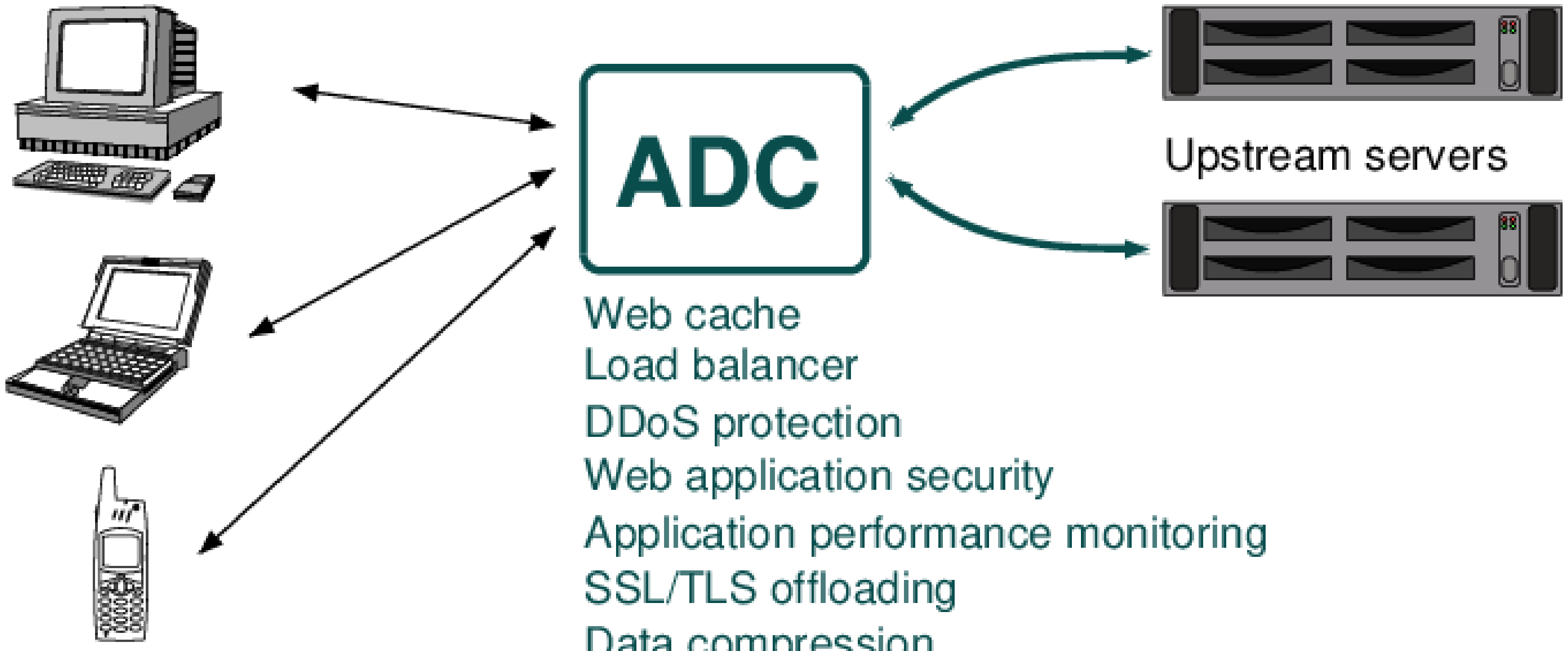


Source: Gartner (July 2015)



Tempesta FW:
Application Delivery Controller (ADC)

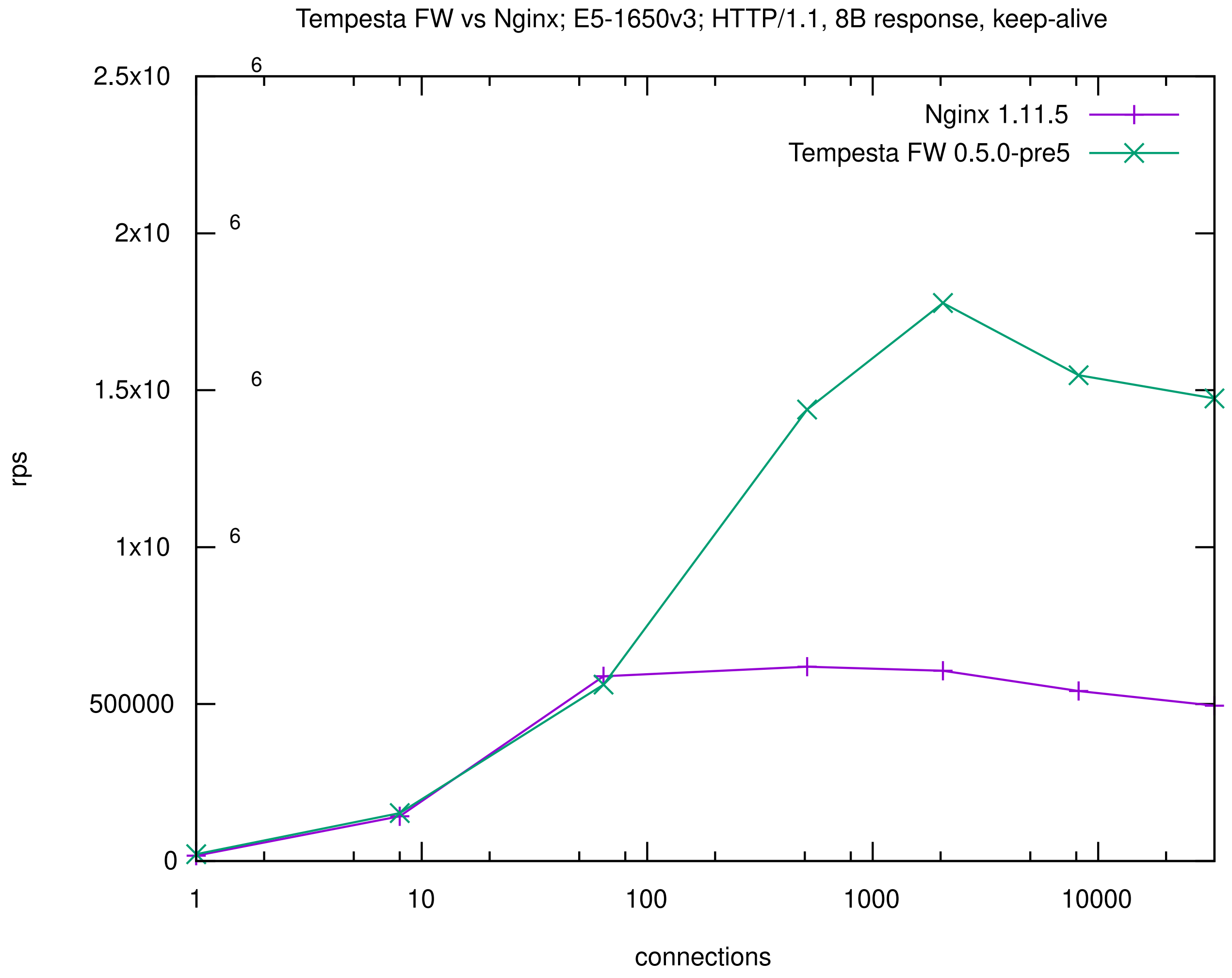
- *https://www.netdevconf.org/2.1/session.html?krizhanovsky*
- **Fast** as kernel bypass, **flexible** as common Linux apps
- HTTP(S) reverse proxy
- **filtering**
  - HTTP DDoS mitigation
  - Web Application Firewall
- built into Linux TCP/IP stack
- up to 1.8M HTTP RPS
  on 4 cores

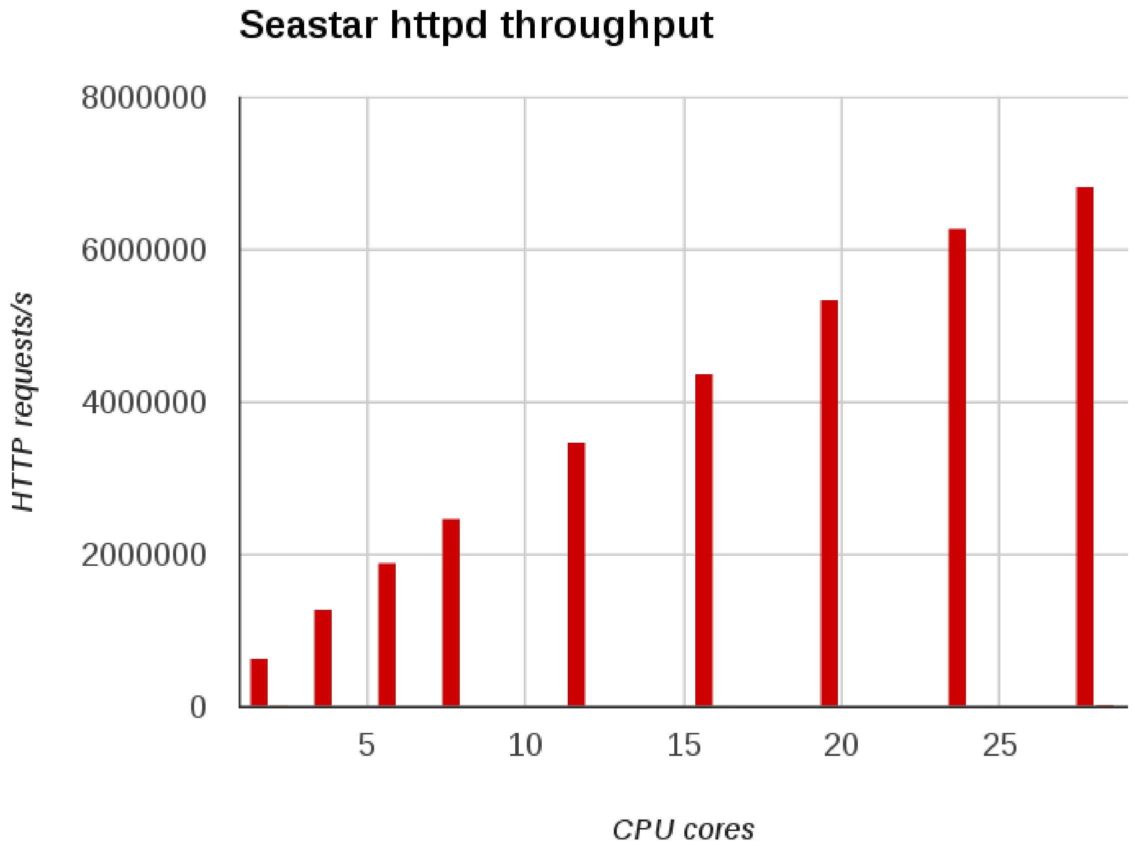


Web cache
Load balancer
DDoS protection
Web application security
Application performance monitoring
SSL/TLS offloading
Data compression
Connections QoS

Tempesta FW performance

- x3 times faster Nginx
- As fast as DPDK-based HTTP Seastar





Problem: HTTP filtration

- 2013: WAF development by request of Positive Technologies
  - Web attacks
  - L7 HTTP/HTTPS DDoS attacks
- Nginx, HAProxy, etc. - perfect HTTP proxies, not HTTP filters
- Netfilter works in TCP/IP stack (softirq) => **HTTP(S)/TCP/IP stack**
- **Tempesta FW**: a hybrid of HTTP accelerator & firewall

- *Disclaimer: Nginx is used just as an example*

HTTP/*(1,~2)* example

```
GET /searchresults.en-us.html?aid=304142&label=gen173nr-342396dbc1b331fab24&tmpl=searchresults&
ac_click_type=b&ac_position=0&checkin_month=3&checkin_monthday=7&checkin_year=2019&checkout_month=3&c
heckout_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_purpose=business&search_selected=1&shw_apar
th=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=\r\n
Host: www.example.com\r\n
Referer: vulnerable.host.net\r\n
Connection: keep-alive\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.116 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
Accept-Encoding: gzip, deflate, sdch\r\n
Accept-Language: en-US,en;q=0.8,ru;q=0.6\r\n
Cookie: a=sdfasd; sdf=3242u389erfhhs; djcnjhe=sdfsdafsdjfb324te1267dd;
sdaf=mo2u8943478t67437461746rfdgfcdc; ityu=9u489573484duifhd; GTYFT=nsdjhcbyq3te76ewgfcZ;
uityut=23Y746756247856425784657; GA=URHUFVHHVSDNFDHGYSDGF; a=%45345%dfdfg
%4656%4534sdfjhsdb.sdfsg.sdfgsf.; aa=4583478; aaaaa=34435345; rrr=iy7t67t6tsdf;
ggg=234i5y24785y78ry534785; sdf=3242u389erfhhs; ityu=9u489573484duifhd; GTYFT=nsdjhcbyq3te76ewgfcZ;
uityut=23Y746756247856425784657; GA=URHUFVHHVSDNFDHGYSDGF; a=%45345%dfdfg
%4656%4534sdfjhsdb.sdfsg.sdfgsf.; nsdjhfb=4358345y; jkbsdff=aaaa; aa=4583478;
ggg=234i5y24785y78ry534785; mmm=23uy47fbhdsfbgh; bsdfhbhfgdqqwew=883476757%345345; jksdfb=2348y;
ndfsgsfdg=235trHHVGHFGC; erertrt=3242342343423324234; g=888888888788\r\n
\r\n
```

HTTP/2 & HTTP/3 (QUIC)
(mix of binary data and strings)

- **Not about saving CPU cycles**
- First occurencies in dynamic table aren’t indexed
- Dynamic table is limited
- HPACK/QPACK is optional
- Huffman
  - Crosses byte bound – can not be vectorized => **very slow**
  - No sense to embed into HTTP parser (conditions for each sub-byte)
- Cookie, User-Agent, Referer, URI can be extremely large
- Cookie and other security sensitive data must not be compressed

Slow HTTP processing

- Dummy **HTTP FSMs**
- **HTTP strings** are special: LIBC functions don’t work well
- HTTP/2 processor typically calls HTTP/1 parsing routines
- Malicious traffic targets the slowest (weakest) point

e.g. Nginx HTTP flood profile

- Whole content is in the cache
- Access log switched off

```
   %              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: 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

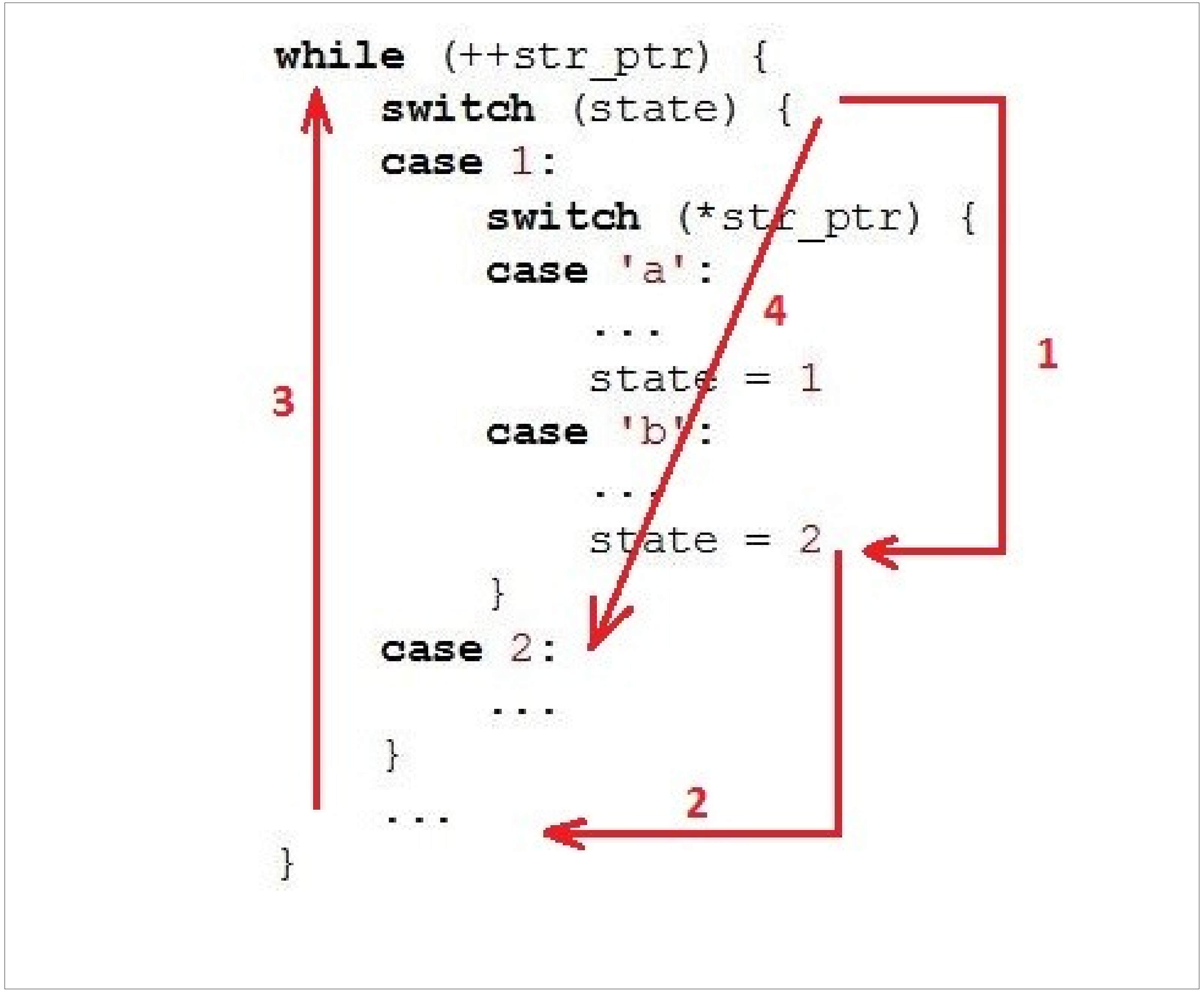
```
Start: state = 1, *str_ptr = 'b'

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

Tempesta
Technologies

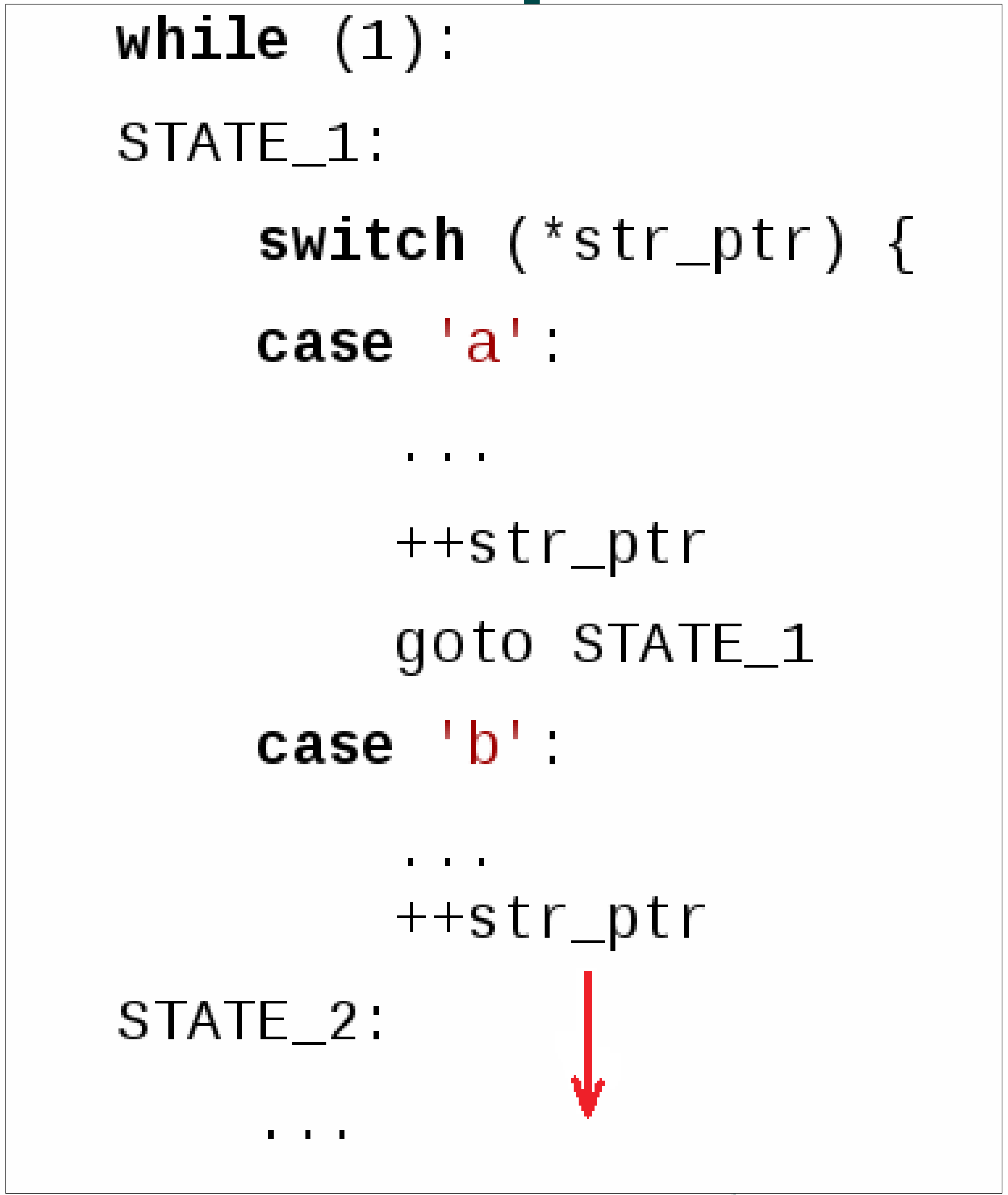
Web-accelerators are slow: HTTP parser

```
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
            }
            ...
        }
```

Web-accelerators are slow: HTTP parser



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



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

ngx_http_parse_request_line()

- Copied I/O – can calculate **token length**
- ‘GET’ is always in one data chunk

```
for (p = b->pos; p < b->last; p++) {
    ...
    switch (state) {
    ...
    case sw_method:
        if (ch == ' ') {
            m = r->request_start;
            switch (p - m) {         // switch on token length!
            case 3:
                if (ngx_str3_cmp(m, 'G', 'E', 'T', ' ')) {
                ...
        }
        if ((ch < 'A' || ch > 'Z') && ch != '_' && ch != '-')
            return NGX_HTTP_PARSE_INVALID_METHOD;
        break;
```

GCC switch optimization: lookup table

```
$ gcc -O2 -S -fverbose-asm -o http_ngx.s http_ngx.c
```

```
ngx_request_line() {
    enum {
        sw_start = 0,
        ...
        sw_almost_done // 26
    } state;

    ...

    switch (state) {
        case sw_start:
        ...
        case sw_almost_done:
        ...
```

```
# switch (state) {
        cmpl    $26, %eax
        ja      .L2309      # end of switch
        jmp     *.L2311(,%rax,8) # <= Spectre!
        ...
.L2311:
        .quad   .L2337      # 0 = sw_start
        ...
        .quad   .L2310      # 26 = sw_almost_done
        ...
.L2337:                     # r->request_start = p;
        movq    %rsi, 96(%rdi)
                      # if (ch == CR || ch == LF) {
        cmpb    $13, %cl
        ...
```

GCC switch optimization: binary search

```
$ gcc -O2 -S -fverbose-asm -o http_ngx.s http_ngx.c
```

```
ngx_request_line() {
    enum {
        sw_start = 0,
        sw_method = 100,
        ...
        sw_http_09 = 215,
        ...
        sw_check_uri = 314,
        ...
        sw_almost_done = 100500
    } state;

    ...

    switch (state) {
        case sw_start:
        ...
        case sw_almost_done:
        ...
```

```
# switch (state) {

        cmpl    $222, %eax
        je      .L2511
        jg      .L2310
        cmpl    $215, %eax
        je      .L2512
        jg      .L2312

        ...

.L2310:
        cmpl    $320, %eax
        je      .L2514
        jg      .L2329
        cmpl    $316, %eax
        je      .L2502
        jg      .L2331
        cmpl    $314, %eax
        jne     .L2641

        ...
```

HTTP parser code size

```
$ nm -S /opt/nginx-1.11.5/sbin/nginx
   | grep http_parse | cut -d' ' -f 2
   | perl -le '$a += hex($_) while (<>); print $a'

9220
```

```
$ getconf LEVEL1_ICACHE_SIZE

32768
```

```
$ grep -c 'case sw_' src/http/ngx_http_parse.c

84
```

- **Tokenization only** in `ngx_http_parse_header_line()`
  (If you need some header value – scan headers table & parse again)
- Web security: **strict header names and values validation**

Tempesta FW: strict HTTP validation

- Zero-copy I/O – **large** HTTP parser becomes the bottleneck
- Zero-copy I/O - ‘GET’ *may (rarely)* come as ‘GE’, ‘T’
  => need to store state between data chunks

```
$ grep -c '__FSM_STATE\|__FSM_TX\|__FSM_METH_MOVE\|__TFW_HTTP_PARSE_' http_parser.c
520
```

```
7.64%  [tempesta_fw]  [k] tfw_http_parse_req
2.79%  [e1000]        [k] e1000_xmit_frame
2.32%  [tempesta_fw]  [k] __tfw_strspn_simd
2.31%  [tempesta_fw]  [k] __tfw_http_msg_add_str_data
1.60%  [tempesta_fw]  [k] __new_pgfrag
1.58%  [kernel]       [k] skb_release_data
1.55%  [tempesta_fw]  [k] __str_grow_tree
1.41%  [kernel]       [k] __inet_lookup_established
1.35%  [tempesta_fw]  [k] tfw_cache_do_action
1.35%  [tempesta_fw]  [k] __tfw_strcmpspn
```

Tempesta
Technologies

Direct jumps FSM

- GOTO and single-names labels give us direct jump FSM
- No auxiliary state variables and updates
- Other examples: Ragel

```
#define FSM_START(s)    switch (s)

#define STATE(st)       case st: st:

// for(;;) body is repeated.
// GCC does very close.
#define MOVE(to, n)                  \
do {                                 \
    p += n;                          \
    if (p > buf + size)              \
        goto done;                   \
    goto to;                         \
} while (0)
```

```
FSM_START(parser->state);

STATE(sw_start) {
    ...
    MOVE(sw_name);
}

STATE(sw_name) {
    ...
```

Replace `switch` by direct jumps

- GCC Labels as values: https://gcc.gnu.org/onlinedocs/gcc/Labels-as-Values.html

```
#define FSM_START(s)    do {            \
    if (!parser->__state)               \
        parser->__state = &&from;\
    goto *parser->__state;              \
} while (0)

#define STATE(st)       st:

// for(;;) body is repeated.
// GCC does very close.
#define MOVE(to, n)                      \
do {                                     \
    p += n;                             \
    if (p > buf + size)                 \
        goto done;                      \
    goto to;                            \
} while (0)
```

```
FSM_START(sw_start);

STATE(sw_start) {
    ...
    MOVE(sw_name);
}

STATE(sw_name) {
    ...
```

Direct jumps vs `switch`: performance

- *https://github.com/tempesta-tech/blog/tree/master/http_benchmark*
- *(`taskset(1)`; Several runs - smallest numbers, not average!)*

```
$ grep -m 2 'model name\|bugs' /proc/cpuinfo
model name : Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz
bugs     : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf

$ gcc --version|head -1
gcc (GCC) 8.2.1 20181105 (Red Hat 8.2.1-5)
```

States	**Switch-driven automaton**		**Goto-driven automaton**	
7	header_line:	**139**ms	header_line:	**156**ms
27	request_line:	**210**ms	request_line:	**186**ms
406	big_header_line:	**1406**ms	goto_big_header_line:	**727**ms

Branch prediction & L1i cache

- perf record -e **branch-misses** -g ./http_benchmark
  - 406 states: switch – **38%** on switch(),
    direct jumps – 13% on header value parsing
  - 7,27 states: switch – <18% switch(), up to 40% for()
    direct jumps – up to **46%** on header & URI parsing

- perf stat -e L1-icache-load-misses ./http_benchmark

	Switch-driven automaton	**Goto-driven automaton**
big FSM code size:	**29156**	49202
L1-icache-load-misses:	4M	**2M**

Tempesta
Technologies

GCC labeled code reordering

```
STATE(sw_method) {
    ... // the most frequent states
    MATCH(NGX_HTTP_GET, "GET ");
    MATCH(NGX_HTTP_POST, "POST");

    ... // many other states

    // Improbable states
    METH_MOVE(Req_MethU, 'N',
              Req_MethUn);
    METH_MOVE(Req_MethUn, 'L',
              Req_MethUnl);
    METH_MOVE(Req_MethUnl, 'O',
              Req_MethUnlo);
    METH_MOVE(Req_MethUnlo, 'C',
              Req_MethUnloc);
    METH_MOVE_finish(Req_MethUnloc, 'K',
                    NGX_HTTP_UNLOCK)
```

```
.L7272:
# http_goto.c:1166: METH_MOVE(Req_MethUnlo,
#                   'C', Req_MethUnloc);
    cmpb    $67, %cl
    jne     .L7362

# ¾ of the function!!!

# http_goto.c:630: MATCH(NGX_HTTP_GET,
#                        "GET ");
    movl    $2, 176(%rdi)
    movl    $4, %eax

# ... some more states

# http_goto.c:635: MATCH(NGX_HTTP_POST,
#                        "POST");
    movl    $8, 176(%rdi)
    movl    $4, %eax
    jmp     .L7354
```

Compiler barrier

- **4%** performance improvement

```
STATE(sw_method) {
    ... // the most frequent states
    MATCH(NGX_HTTP_GET, "GET ");
    MATCH(NGX_HTTP_POST, "POST");

    __asm__ __volatile__("": : :"memory");

    ... // many other states

    // Improbable states
    METH_MOVE(Req_MethU, 'N', Req_MethUn);
    METH_MOVE(Req_MethUn, 'L', Req_MethUnl);
    METH_MOVE(Req_MethUnl, 'O', Req_MethUnlo);
    METH_MOVE(Req_MethUnlo, 'C', Req_MethUnloc);
    METH_MOVE_finish(Req_MethUnloc, 'K', NGX_HTTP_UNLOCK)
```

Towards better code layout

- Profiler guided optimization (**PGO**) – total samples, not call sequence (ex. URI gets more samples, so comes before method parsing)
- `hot`/`cold` label attributes & `likely`/`unlikely` hints
  - Compiler barrier is fine with branch optimizations
  - `likely` moves labeled code into `if`
  - `hot`/`cold` move labeled code up/below

```
Req_Method: {
    if (likely(PI(p) == CHAR4_INT('G', 'E', 'T', ' '))) {
        ...
        goto Req_Uri;
    }
    if (likely(PI(p) == CHAR4_INT('P', 'O', 'S', 'T'))) {
        ...
        goto Req_UriSpace;
    }
    goto Req_Meth_SlowPath;
}
... // other methods: POST, PUT etc.

Req_Uri:
    ... // URI processing

Req_Meth_SlowPath:
    ...
```

Towards better code layout

- Profiler guided optimization (**PGO**) – total samples, not call sequence (ex. URI gets more samples, so comes before method parsing)
- `hot/cold` label attributes & `likely/unlikely` hints
  - Compiler barrier is fine with branch optimizations
  - `likely` moves labeled code into `if`
  - `hot/cold` move labeled code up/below

```
Req_Method: {
    if (likely(PI(p) == CHAR4_INT('G', 'E', 'T', ' '))) {
        ...
        goto Req_Uri;
    }
    if (PI(p) == CHAR4_INT('P', 'O', 'S', 'T')) {
        ...
        goto Req_UriSpace;
    }
    goto Req_Meth_SlowPath;
}
... // other methods: POST, PUT etc.

Req_Uri:__attribute__((hot))
    ... // URI processing

Req_Meth_SlowPath:__attribute__((cold))
    ...
```

Ambiguous -O3

```
$ for i in `seq 1 3`; do time taskset 0x2 ./http_benchmark; done
```

```
-O2:                             1.838s
-O3:                             1.858s
-finline-functions               1.832s
-funswitch-loops                 1.830s
-fpredictive-commoning           1.853s
-fgcse-after-reload              1.832s
-ftree-loop-vectorize            1.868s
-ftree-loop-distribution         1.839s
-ftree-loop-distribute-patterns  1.842s
-floop-interchange               1.823s
-floop-unroll-and-jam            1.835s
-fsplit-paths                    1.834s
-ftree-slp-vectorize             1.837s
-fvect-cost-model                1.846s
-ftree-partial-pre               1.842s
-fpeel-loops                     1.827s
-fipa-cp-clone                   1.822s
-O2 -floop-interchange -fpeel-loops -fipa-cp-clone 1.820s
```

Auto-vectorization

- Enabled on `-O3`
- `-fopt-info-vec-all` shows what is optimized
- Not everything is vectorizable:

```
$ gcc -O3 -ftree-vectorizer-verbose=2 -fopt-info-vec -c *.c 2>&1 |wc -l
0
```

- Auto-vectorization in GCC, *https://www.gnu.org/software/gcc/projects/tree-ssa/vectorization.html*

```
int a[256], b[256], c[256];
void foo () {
    for (int i = 0; i < 256; i++)
        a[i] = b[i] + c[i];
}
```

Alignment: how to match GET?

- How it's expensive if p isn't aligned?

```
#define CHAR4_INT(a, b, c, d)    ((d << 24) | (c << 16) | (b << 8) | a)

if (p == CHAR4_INT('G', 'E', 'T', ' ')))
    // we have GET as method
```

- *https://github.com/tempesta-tech/blog/tree/master/int_align*

```
$ ./int_align
Unaligned access = 6.20482
Aligned access = 2.87012
Read four bytes = 2.45249
```

- Checked access is good enough (*but GCC doesn't agree*)

```
(((long)(p) & 3)
 ? ((unsigned int)((p)[0]) | ((unsigned int)((p)[1]) << 8)
   | ((unsigned int)((p)[2]) << 16) | ((unsigned int)((p)[3]) << 24))
 : *(unsigned int *)(p));
```

Let’s try this in the parser

- **Results:**

full request line:	**no difference**	
method only:	unaligned	- **214**ms
	aligned	- **231**ms
	bytes	- 216ms

- Why (*https://github.com/tempesta-tech/tempesta/issues/695*)?
  - Compiler optimizations: `p` is read in many places
  - Microbenchmark: minimize optimizations
  - The more complex code confuses the compiler

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;
    ...
```

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
```

What makes HTTP strings special

- (HTTP/1) Special delimiters: `‘:’`, `‘,’` or even 2-byte `CRLF`,...
- ...which can be 1-byte `LF` by RFC 7230 recommendation
- No `‘\0’`-termination (if you’re zero-copy)
- **Security:** RFC defines strict alphabets for each HTTP message part
- `strspn()`: limited number of accepted alphabets
- `strspn()` compiles allowed character set in run-time
- `strcasecmp()`: no need case conversion to compare `x` with `‘Foo:’`
- In most cases only `match/not-match` required from `strcasecmp()`
- `switch()`-driven FSM matchers are even worse

URI (RFC 3986) parsing in the wild

- Nginx
  - Switch-driven parser
  - Strict validation of the RFC-defined characters set
- PicoHTTPParser (H2O)
  - Just basic check `0x20 (Space) < ch < 0x7f (DEL)`
  - SSE 4.2 `PCMESTRI` – 16 chars at once
    - only 8 ranges or 16 chars – too smal for URI alphabet
- Cloudflare PicoHTTPParser AVX2 extension
  - Check for `(c >= 0x20 || c == '\t') && c < 0x7f`
  - 32 chars at once

PCMESTRI

```
static const unsigned char ranges[] __attribute__((aligned(16))) =
    "\x00 "         /* control chars and up to SP */
    "\"\""          /* 0x22 */
    "<<"            /* 0x3c,0x3c */
    ">>"            /* 0x3e,0x3e */
    "\\\\"          /* 0x5c,0x5c */
    "^^"            /* 0x5e,0x5e */
    "{}"            /* 0x7b-0x7d */
    "\x7f\xff";    /* 0x7f-0xff */

__m128i ranges16 = _mm_loadu_si128((const __m128i *)ranges);

__m128i b16 = _mm_loadu_si128((void *)s);

int r = _mm_cmpestri(ranges16, ranges_sz, b16, 16,
                     _SIDD_LEAST_SIGNIFICANT | _SIDD_CMP_RANGES | _SIDD_UBYTE_OPS);
```

AVX2 (CloudFlare’s approach)

```
const __m256i lb = _mm256_set1_epi8(0x1f); /* low bound */
const __m256i hb = _mm256_set1_epi8(0x7f); /* high bound */
const __m256i tab = _mm256_set1_epi8(0x09); /* allow TAB */


/* SPACE <= v */
__m256i low = _mm256_cmpgt_epi8(v, lb);

/* SPACE <= v < 0x7f */
__m256i bounds = _mm256_and_si256(_mm256_cmpgt_epi8(hb, v), low);

/* SPACE <= v < 0x7f || v == TAB */
__m256i r = _mm256_or_si256(_mm256_cmpeq_epi8(tab, v), bounds);

/* Generate bit mask */
*range = ~_mm256_movemask_epi8(r);
```

PCMESTRI vs AVX2

PCMPESTRI/PicoHTTPParser:

```
str_len    1:     128ms
str_len    3:     138ms
str_len   10:     161ms
str_len   19:     151ms
str_len   28:     183ms
str_len  107:     218ms
str_len  178:     230ms
str_len 1023:     784ms
str_len 1500:    1069ms
```

AVX2/CloudFlare:

```
str_len    1:     171ms
str_len    3:     175ms
str_len   10:     189ms
str_len   19:     174ms
str_len   28:     196ms
str_len  107:     198ms
str_len  178:     203ms
str_len 1023:     375ms
str_len 1500:     458ms
```

If you’re thinking about `strspn(3)`...

```
                                  GLIBC strspn():

str_len     1:     128ms     171ms     350ms
str_len     3:     138ms     175ms     354ms
str_len    10:     161ms     189ms     380ms
str_len    19:     151ms     174ms     420ms
str_len    28:     183ms     196ms     398ms
str_len   107:     218ms     198ms     533ms
str_len   178:     230ms     203ms     650ms
str_len  1023:     784ms     375ms    2071ms
str_len  1500:    1069ms     458ms    2856ms
```

Tempesta matcher: even faster and accurate

Tempesta
AVX2 constant URI matching

str_len 1:	**128**ms	**171**ms	**123**ms
str_len 3:	**138**ms	**175**ms	**127**ms
str_len 10:	**161**ms	**189**ms	**150**ms
str_len 19:	**151**ms	**174**ms	**139**ms
str_len 28:	**183**ms	**196**ms	**156**ms
str_len 107:	**218**ms	**198**ms	**167**ms
str_len 178:	**230**ms	**203**ms	**180**ms
str_len 1023:	**784**ms	**375**ms	**350**ms
str_len **1500**:	**1069**ms	**458**ms	**433**ms

Short strings

```
static const unsigned char uri_a[] __attribute__((aligned(64))) = {
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        ...

        // Branch misprediction is more crucial for short strings
        if (likely(len <= 4)) {
                switch (len) {
                case 0:
                        return 0;
                case 4:
                        c3 = uri_a[s[3]];
                        // fall through to process other chars
                case 3:
                        c2 = uri_a[s[2]];
                case 2:
                        c1 = uri_a[s[1]];
                case 1:
                        c0 = uri_a[s[0]];
                }
                return (c0 & c1) == 0 ? c0 : 2 + (c2 ? c2 + c3 : 0);
        }
```

Main loop & large tail

```
for ( ; unlikely(s + 128 <= end); s += 128) {
        n = match_symbols_mask128_c(__C.URI_BM, s);
        if (n < 128)
                return s - (unsigned char *)str + n;
}
if (unlikely(s + 64 <= end)) {
        n = match_symbols_mask64_c(__C.URI_BM, s);
        if (n < 64)
                return s - (unsigned char *)str + n;
        s += 64;
}
if (unlikely(s + 32 <= end)) {
        n = match_symbols_mask32_c(__C.URI_BM, s);
        if (n < 32)
                return s - (unsigned char *)str + n;
        s += 32;
}
if (unlikely(s + 16 <= end)) {
        n = match_symbols_mask16_c(__C.URI_BM128, s);
        if (n < 16)
                return s - (unsigned char *)str + n;
        s += 16;
}
```

Tail

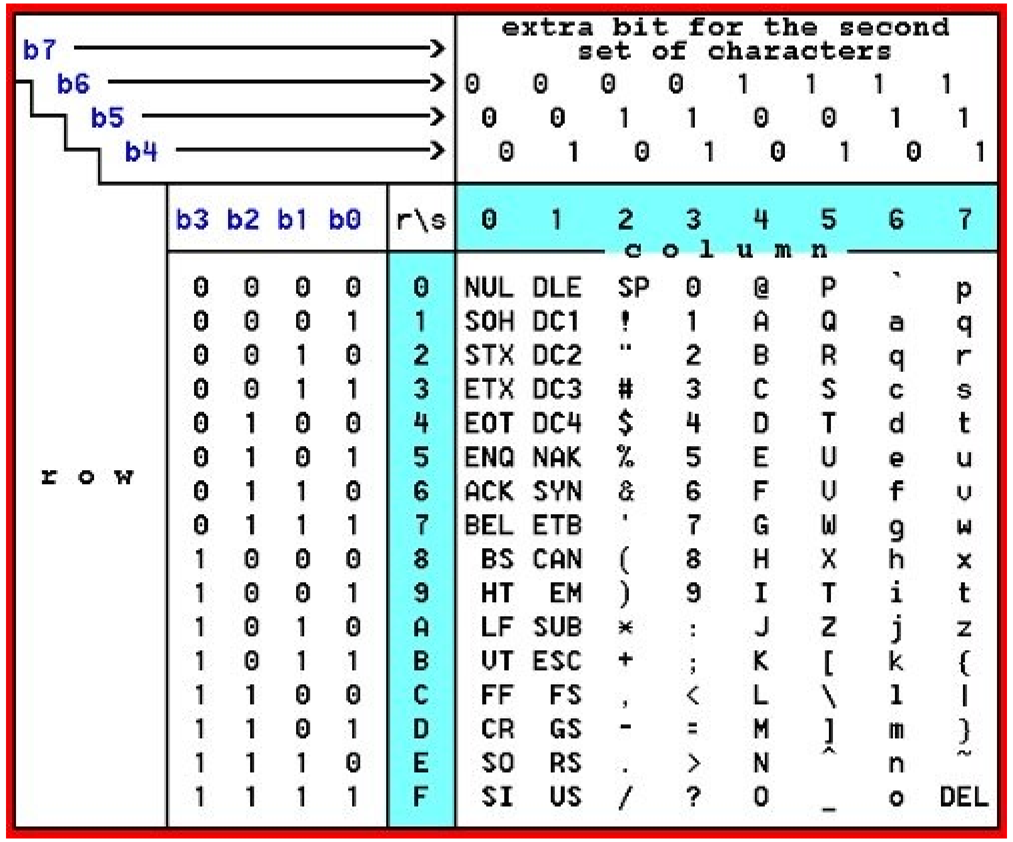
```
while (s + 4 <= end) {
        c0 = uri_a[s[0]];
        c1 = uri_a[s[1]];
        c2 = uri_a[s[2]];
        c3 = uri_a[s[3]];
        if (!(c0 & c1 & c2 & c3)) {
                n = s - (unsigned char *)str;
                return !(c0 & c1) ? n + c0 : n + 2 + (c2 ? c2 + c3 : 0);
        }
        s += 4;
}
c0 = c1 = c2 = 0;
switch (end - s) {
case 3:
        c2 = uri_a[s[2]];
case 2:
        c1 = uri_a[s[1]];
case 1:
        c0 = uri_a[s[0]];
}
n = s - (unsigned char *)str;
return !(c0 & c1) ? n + c0 : n + 2 + c2;
```

Load bitmask & data

```
const __m256i ARF = _mm256_setr_epi8(
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0,
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0);
URI_BM = _mm256_setr_epi8(
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c,
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c);
const __m256i LSH = _mm256_set1_epi8(0xf);

__m256i v = _mm256_lddqu_si256((void *)str);

__m256i acbm = _mm256_shuffle_epi8(URI_BM, v);

__m256i acols = _mm256_and_si256(LSH,
                           _mm256_srli_epi16(v, 4));

__m256i arbits = _mm256_shuffle_epi8(ARF, acols);

__m256i sbits = _mm256_and_si256(arbits, acbm);

v = _mm256_cmpeq_epi8(sbits, _mm256_setzero_si256());

return __tzcnt(0xffffffff00000000UL
               | _mm256_movemask_epi8(v));
```

b8 = inv(1011 1000) = 0 @ P p

f8 = inv(1111 1000) = 2 B R q R

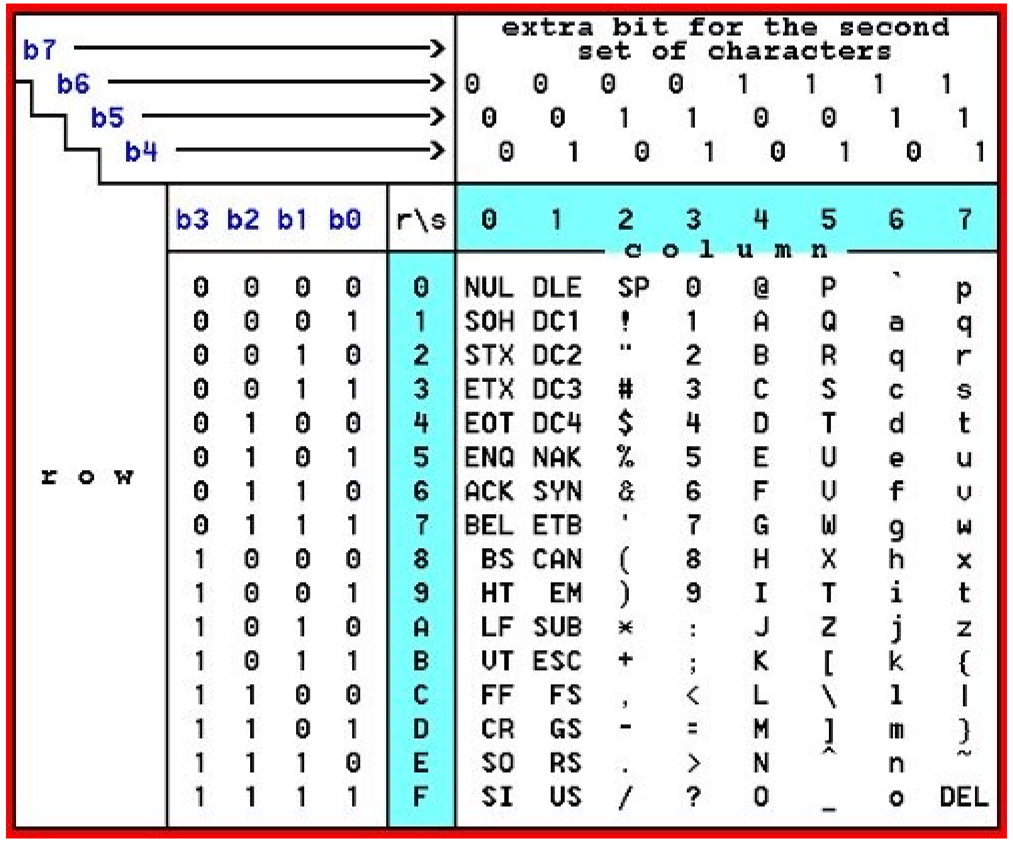


b7					extra bit for the second set of characters							
b6					0	0	0	0	1	1	1	1
b5					0	0	1	1	0	0	1	1
b4					0	1	0	1	0	1	0	1
	b3	b2	b1	b0	r\s	0	1	2	3	4	5	6	7
						column							
row	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	q	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	U	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	T	i	t
	1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	B	UT	ESC	+	;	K	[	k	{
	1	1	0	0	C	FF	FS	,	<	L	\	l	|
	1	1	0	1	D	CR	GS	-	=	M	]	m	}
	1	1	1	0	E	SO	RS	.	>	N	^	n	~
	1	1	1	1	F	SI	US	/	?	O	_	o	DEL

Arrange ASCII row bitmasks

```
const __m256i ARF = _mm256_setr_epi8(
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0,
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0);
URI_BM = _mm256_setr_epi8(
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c,
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c);
const __m256i LSH = _mm256_set1_epi8(0xf);

__m256i v = _mm256_lddqu_si256((void *)str);

__m256i acbm = _mm256_shuffle_epi8(URI_BM, v);

__m256i acols = _mm256_and_si256(LSH,
                          _mm256_srli_epi16(v, 4));

__m256i arbits = _mm256_shuffle_epi8(ARF, acols);

__m256i sbits = _mm256_and_si256(arbits, acbm);

v = _mm256_cmpeq_epi8(sbits, _mm256_setzero_si256());

return __tzcnt(0xffffffff00000000UL
               | _mm256_movemask_epi8(v));
```

```
str="pr":    p = 0x70
             r = 0x72
```



						extra bit for the second set of characters							
b7													
b6						0	0	0	0	1	1	1	1
b5						0	0	1	1	0	0	1	1
b4						0	1	0	1	0	1	0	1
	b3	b2	b1	b0	r\s	0	1	2	3	4	5	6	7
						column							
row	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	q	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	T	i	t
	1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	B	UT	ESC	+	;	K	[	k	{
	1	1	0	0	C	FF	FS	,	<	L	\	l	|
	1	1	0	1	D	CR	GS	-	=	M	]	m	}
	1	1	1	0	E	SO	RS	.	>	N	^	n	~
	1	1	1	1	F	SI	US	/	?	O	_	o	DEL

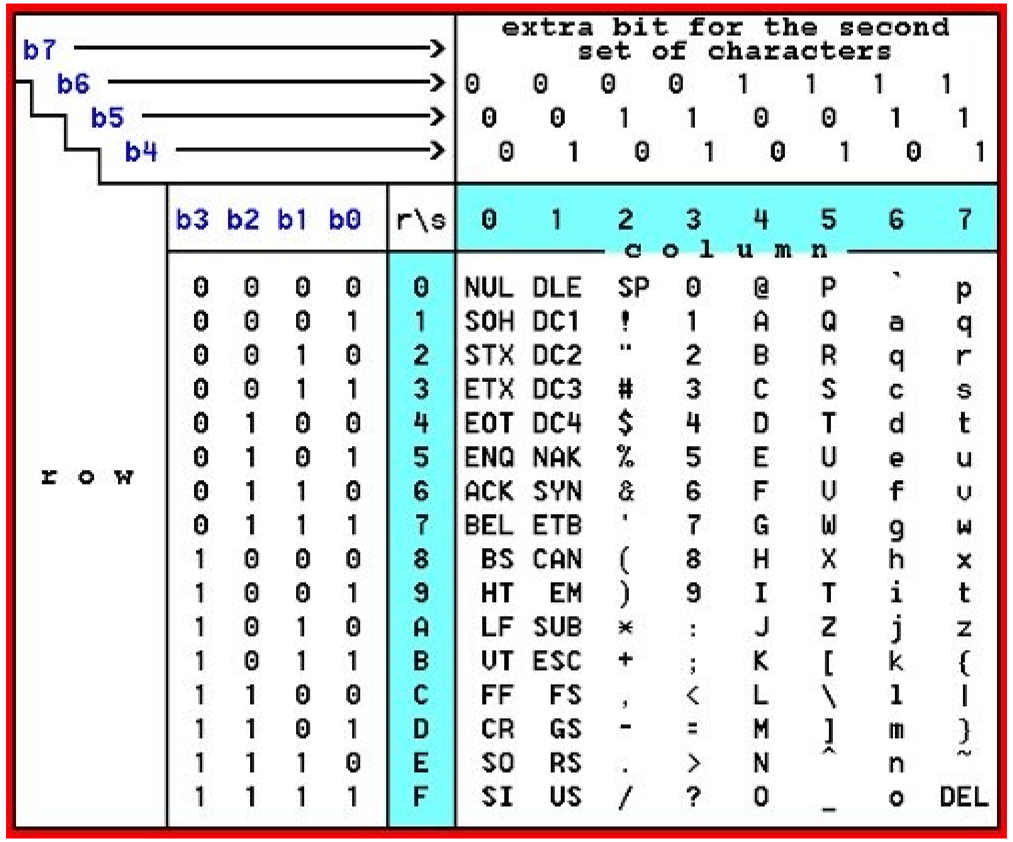
Tempesta Technologies

Get column IDs for the input data

```
const __m256i ARF = _mm256_setr_epi8(
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0,
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0);
URI_BM = _mm256_setr_epi8(
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c,
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c);
const __m256i LSH = _mm256_set1_epi8(0xf);

__m256i v = _mm256_lddqu_si256((void *)str);

__m256i acbm = _mm256_shuffle_epi8(URI_BM, v);

__m256i acols = _mm256_and_si256(LSH,
                          _mm256_srli_epi16(v, 4));

__m256i arbits = _mm256_shuffle_epi8(ARF, acols);

__m256i sbits = _mm256_and_si256(arbits, acbm);

v = _mm256_cmpeq_epi8(sbits, _mm256_setzero_si256());

return __tzcnt(0xffffffff00000000UL
               | _mm256_movemask_epi8(v));
```

pr = 0x70 0x72 >> 4

7th column: 0x0707 (16bits)

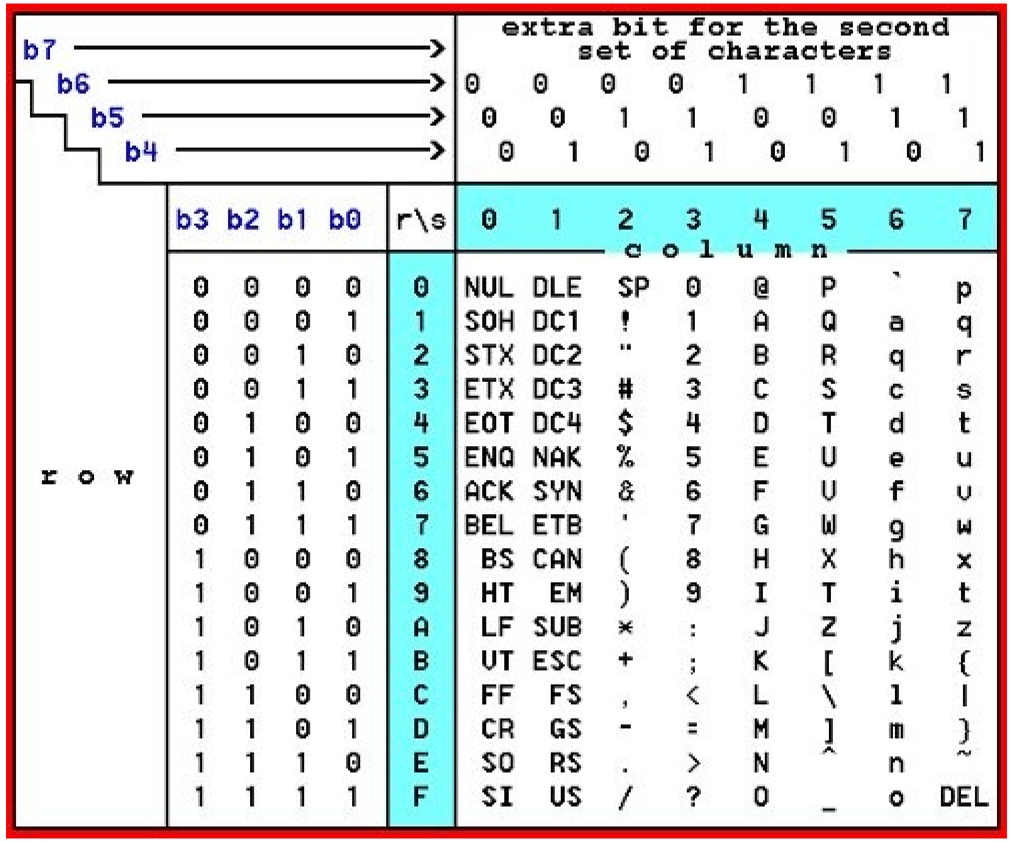


b7						extra bit for the second set of characters							
b6						0	0	0	0	1	1	1	1
b5						0	0	1	1	0	0	1	1
b4						0	1	0	1	0	1	0	1
	b3	b2	b1	b0	r\s	0	1	2	3	4	5	6	7
						column							
row	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	q	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	U	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	T	i	t
	1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	B	VT	ESC	+	;	K	[	k	{
	1	1	0	0	C	FF	FS	,	<	L	\	l	|
	1	1	0	1	D	CR	GS	-	=	M	]	m	}
	1	1	1	0	E	SO	RS	.	>	N	^	n	~
	1	1	1	1	F	SI	US	/	?	O	_	o	DEL

Arrange ASCII columns

```
const __m256i ARF = _mm256_setr_epi8(
   0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0,
   0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0);
URI_BM = _mm256_setr_epi8(
   0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
   0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c,
   0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
   0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c);
const __m256i LSH = _mm256_set1_epi8(0xf);

__m256i v = _mm256_lddqu_si256((void *)str);

__m256i acbm = _mm256_shuffle_epi8(URI_BM, v);

__m256i acols = _mm256_and_si256(LSH,
                        _mm256_srli_epi16(v, 4));

__m256i arbits = _mm256_shuffle_epi8(ARF, acols);

__m256i sbits = _mm256_and_si256(arbits, acbm);

v = _mm256_cmpeq_epi8(sbits, _mm256_setzero_si256());

return __tzcnt(0xffffffff00000000UL
               | _mm256_movemask_epi8(v));
```

```
rows:     0xb8 0xf8
columns:  0x80 0x80
```



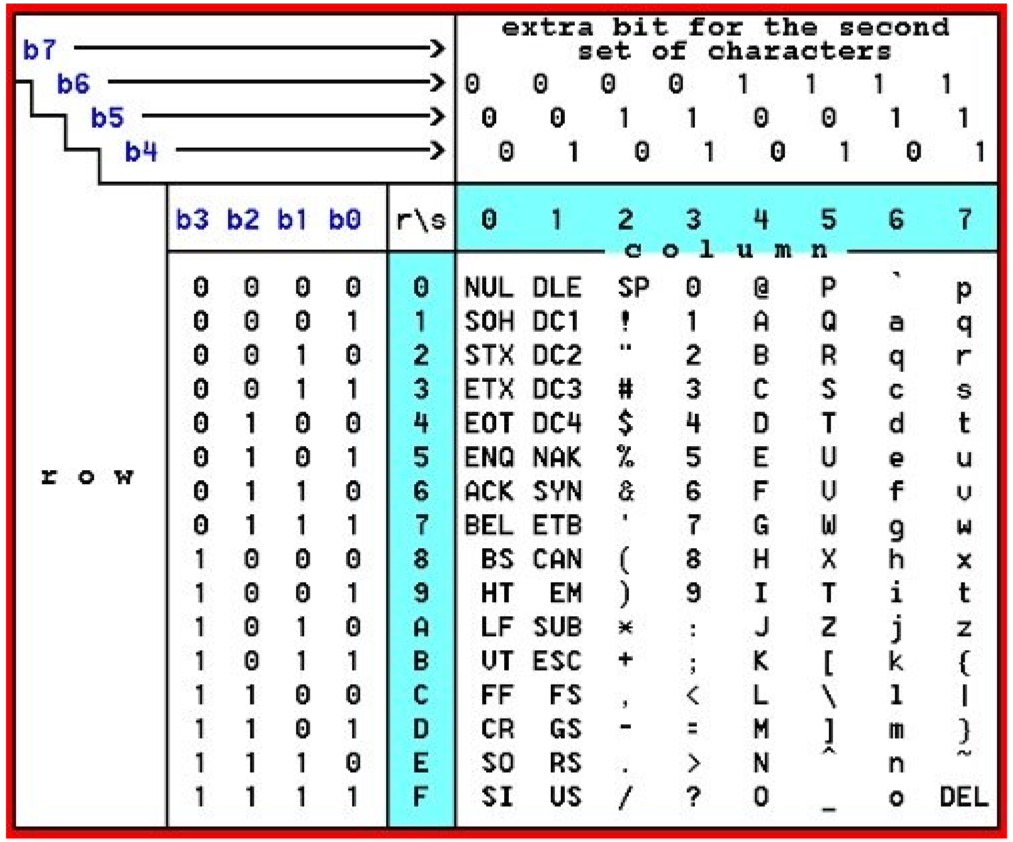
						extra bit for the second set of characters							
b7													
b6						0	0	0	0	1	1	1	1
b5						0	0	1	1	0	0	1	1
b4						0	1	0	1	0	1	0	1
	b3	b2	b1	b0	r\s	0	1	2	3	4	5	6	7
						column							
row	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	q	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	T	i	t
	1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	B	VT	ESC	+	;	K	[	k	{
	1	1	0	0	C	FF	FS	,	<	L	\	l	|
	1	1	0	1	D	CR	GS	-	=	M	]	m	}
	1	1	1	0	E	SO	RS	.	>	N	^	n	~
	1	1	1	1	F	SI	US	/	?	O	_	o	DEL

Tempesta
Technologies

Intersect columns and rows bitmasks

```
const __m256i ARF = _mm256_setr_epi8(
    0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0,
    0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0);
URI_BM = _mm256_setr_epi8(
    0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
    0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c,
    0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
    0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c);
const __m256i LSH = _mm256_set1_epi8(0xf);

__m256i v = _mm256_lddqu_si256((void *)str);

__m256i acbm = _mm256_shuffle_epi8(URI_BM, v);

__m256i acols = _mm256_and_si256(LSH,
                          _mm256_srli_epi16(v, 4));

__m256i arbits = _mm256_shuffle_epi8(ARF, acols);

__m256i sbits = _mm256_and_si256(arbits, acbm);

v = _mm256_cmpeq_epi8(sbits, _mm256_setzero_si256());

return __tzcnt(0xffffffff00000000UL
               | _mm256_movemask_epi8(v));
```

“pr” is allowed
0xb8f8 & 0x8080 = 0x8080

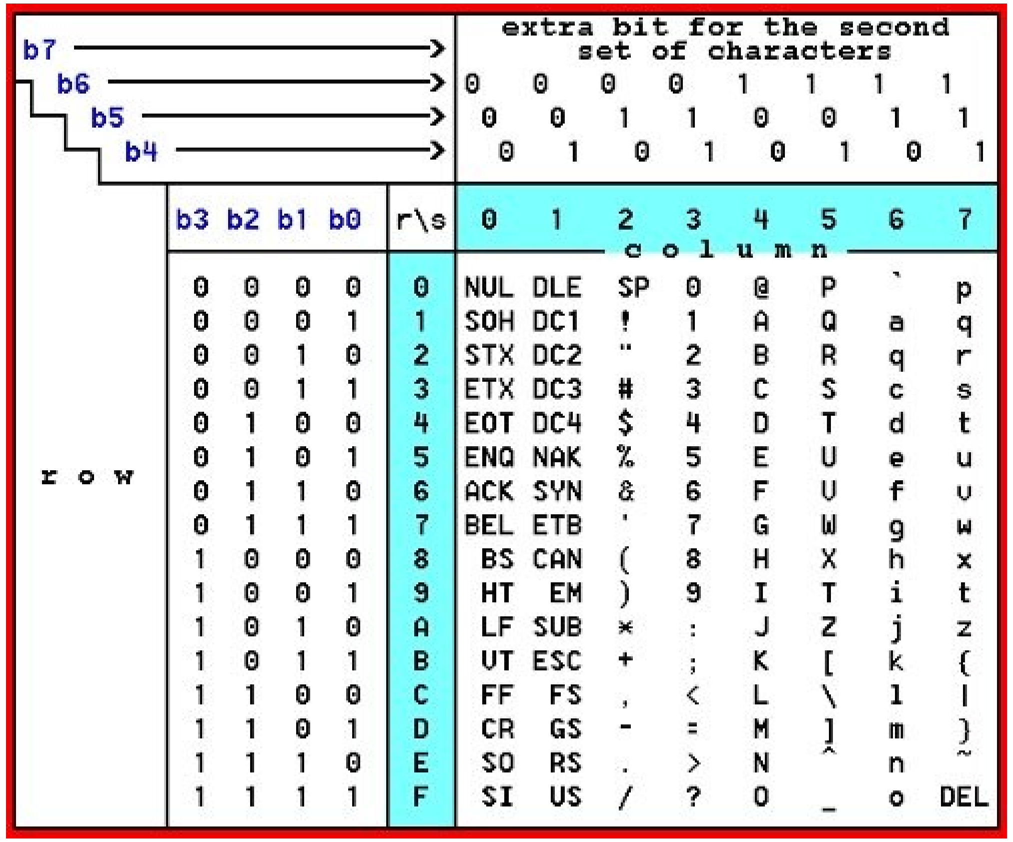


extra bit for the second set of characters

b7												
b6						0	0	0	0	1	1	1	1
b5						0	0	1	1	0	0	1	1
b4						0	1	0	1	0	1	0	1
	b3	b2	b1	b0	r\s	0	1	2	3	4	5	6	7
						column							
row	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	q	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	U	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	T	i	t
	1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	B	VT	ESC	+	;	K	[	k	{
	1	1	0	0	C	FF	FS	,	<	L	\	l	|
	1	1	0	1	D	CR	GS	-	=	M	]	m	}
	1	1	1	0	E	SO	RS	.	>	N	^	n	~
	1	1	1	1	F	SI	US	/	?	O	_	o	DEL

Count non-zero bytes

```
const __m256i ARF = _mm256_setr_epi8(
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0,
  0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0, 0, 0, 0, 0, 0, 0, 0);
URI_BM = _mm256_setr_epi8(
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c,
  0xb8, 0xfc, 0xf8, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc,
  0xfc, 0xfc, 0xfc, 0x7c, 0x54, 0x7c, 0xd4, 0x7c);
const __m256i LSH = _mm256_set1_epi8(0xf);

__m256i v = _mm256_lddqu_si256((void *)str);

__m256i acbm = _mm256_shuffle_epi8(URI_BM, v);

__m256i acols = _mm256_and_si256(LSH,
                          _mm256_srli_epi16(v, 4));

__m256i arbits = _mm256_shuffle_epi8(ARF, acols);

__m256i sbits = _mm256_and_si256(arbits, acbm);

v = _mm256_cmpeq_epi8(sbits, _mm256_setzero_si256());

return __tzcnt(0xffffffff00000000UL
               | _mm256_movemask_epi8(v));
```



					extra bit for the second set of characters							
b7					0	0	0	0	1	1	1	1
b6					0	0	1	1	0	0	1	1
b5					0	1	0	1	0	1	0	1
b4												
b3	b2	b1	b0	r\s	0	1	2	3	4	5	6	7
					column							
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	q	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	T	i	t
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	|
1	1	0	1	D	CR	GS	-	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	F	SI	US	/	?	O	_	o	DEL

Even flexible: custom allowed alphabets

- **Custom BM tables to filter injection attacks**
- **SSRF**, “A New Era of SSRF”, BlackHat’17
  - `http://foo@evil.com:80@google.com/`
  - Allow only `[a-zA-Z0-9:/%]` in URI
    ```
    http_uri_brange 0x61-0x7a 0x41-0x5a 0x30-0x3a 0x2f 0x25;
    ```
- **RCE**, “Perform effective command injection attacks like”, BSides’16
  - `User-Agent: ...;echo NAELBD$((26+58))$echo(echo NAELBD)NAELBD...`
  - Allow only `[a-zA-Z0-9;,()/.]` in ctext | VCHAR headers
    ```
    http_ctext_vchar_brange 0x61-0x7a 0x41-0x5a 0x3b 0x2c
                            0x28 0x29 0x2f 0x2e;
    ```
- **Relative Path Overwrite** (Google, 2016):
  `.../gallery?q=%0a{}*{background:red}/..//apis/howto_guide.html`

strcasecmp()

- One of the srings is always in low case
- Similar approach for short strings and tail

```
__m256i CASE = _mm256_set1_epi8(0x20);

// Hacker’s Delight for signed comparison: -0x80 for both operands
__m256i A = _mm256_set1_epi8('A' - 0x80);
__m256i D = _mm256_set1_epi8('Z' - 'A' + 1 - 0x80);

// Hacker’s Delight: 'a' <= v <= 'z' to
// v - ('a' - 0x80) < 'z' - 'a' + 1 - 0x80
__m256i sub = _mm256_sub_epi8(str1, A);
__m256i cmp_r = _mm256_cmpgt_epi8(D, sub);

__m256i lc = _mm256_and_si256(cmp_r, CASE);

__m256i vl = _mm256_or_si256(str1, lc);

__m256i eq = _mm256_cmpeq_epi8(vl, str2);

return ~_mm256_movemask_epi8(eq);
```

strcasecmp() performance

GLIBC:

```
str_len    1:    133ms
str_len    3:    144ms
str_len   10:    143ms
str_len   19:    163ms
str_len   28:    168ms
str_len  107:    213ms
str_len  178:    253ms
str_len 1023:    861ms
str_len 1500:   1167ms
```

Tempesta:

```
str_len    1:    126ms
str_len    3:    129ms
str_len   10:    129ms
str_len   19:    133ms
str_len   28:    136ms
str_len  107:    154ms
str_len  178:    179ms
str_len 1023:    310ms
str_len 1500:    376ms
```

Tempesta
Technologies

FPU in the Linux kernel

- The kernel doesn’t save/restore FPU state on kernel/user-space context switches
- Some code (e.g. crypto) uses SIMD
  => `kernel_fpu_begin()`/`kernel_fpu_end()`
- Tempest FW (TCP/IP stack) works in softirq
  => store/restore FPU context on start/exit softirq handler

```
__kernel_fpu_begin_bh();
memcpy_avx(dst, src, n);
__kernel_fpu_end_bh();
```

Mixing AVX code with SSE code

- AVX to SSE transition penalty
  - use `vzeroupper` instruction
  - convert SSE instructions to AVX (`−msse2avx` or `−mavx` for GCC)
- Skylake behaves better in several AVX to SSE transitions
- GCC in `−O2` generates
  - VEX-prefixed AVX versions of SSE instructions
  - `vzeroupper` before `ret` (**5% performance degradation** on Skylake)
- There are no SSE 3rd-party users in kernel => no need `vzeroupper`

FPU save/restore price (`memcpy()`)

- Hello to user-space auto-vectorization :)
- *https://github.com/tempesta-tech/blog/blob/master/kstrings/memcpy_res.kernel*

Raw:

```
unaligned:      304ms
        8:      110ms
       20:      153ms
       64:      145ms
      120:      181ms
      256:      210ms
      320:      237ms
      512:      309ms
      850:      464ms
     1500:      358ms
```

Safe:

```
unaligned:      1591ms
        8:      2865ms
       20:      2887ms
       64:      2887ms
      120:      2864ms
      256:      3011ms
      320:      3034ms
      512:      3113ms
      850:      3278ms
     1500:      1710ms
```

Intelpocalypse: performance cost

- Spectre (exploiting speculative execution):
  retpoline (*https://support.google.com/faqs/answer/7625886*)
  - ~**15%** perf degradation (*https://github.com/tempesta-tech/tempesta/pull/1249#discussion_r284860003)*

```
jmp *%r11
```

```
    call l1
l0: pause
    lfence
    jmp l0
l1: mov %r11, (%rsp)
    ret
```

- Meltdown (reading kernel memory from user space):
  `CONFIG_PAGE_TABLE_ISOLATION` **(KPTI)**
  - no lazy TLB as previously, PCID instead
  - ~**30%** perf degradation (*one more profit to be in kernel :)*)
    (*https://mariadb.org/myisam-table-scan-performance-kpti/*)

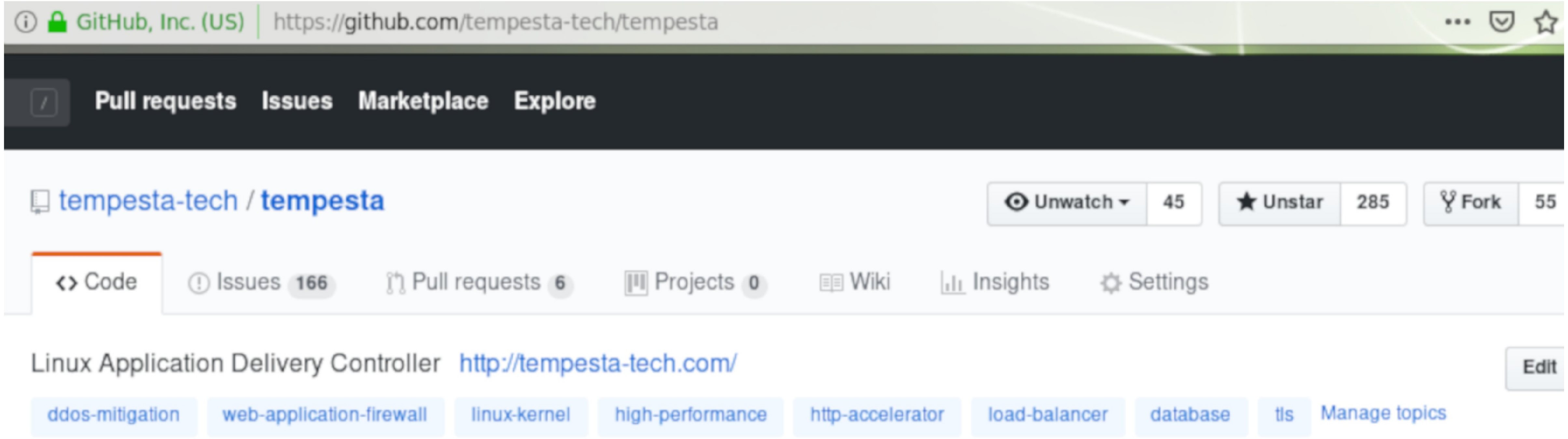
Tempesta
Technologies

References

- “Fast Finite State Machine for HTTP Parsing”, *http://natsys-lab.blogspot.ru/2014/11/the-fast-finite-state-machine-for-http.html*
- “HTTP Strings Processing Using C, SSE4.2 and AVX2”, *http://natsys-lab.blogspot.ru/2016/10/http-strings-processing-using-c-sse42.html*
- Cloudflare, “Improving PicoHTTPParser further with AVX2” *https://blog.cloudflare.com/improving-picohttpparser-further-with-avx2/*
- Hacker’s Delight, *http://www.hackersdelight.org/*
- Intel 64 and IA-32 Architectires Optimization Reference Manual
- Meltdown and Spectre attacks docs, *https://spectreattack.com/*
- Intelpocalypse: goodbye fast system calls, *https://medium.com/@krizhanovsky/intelpocalypse-goodbye-fast-system-calls-a0824683d67e*

Thanks!

- E-mail: *ak@tempesta-tech.com*
- Web-site: *http://tempesta-tech.com*
- Custom software development: *http://tempesta-tech.com/services#custom*



Linux Application Delivery Controller http://tempesta-tech.com/

Tempesta
Technologies