C10M: radical parsers

by Robert David Graham
@ErrataRob
http://c10m.robertgraham.com/
The graph illustrates the relationship between concurrent connections and requests per second. The x-axis represents the number of concurrent connections, while the y-axis shows the percentage of requests per second. The performance and scalability of a system are indicated by the curves, which show how the system's ability decreases as the number of concurrent connections increases.
pointer chasing

• At scale (millions of connects) nothing is in high-speed cache
• Packet-descriptors -> TCB -> socket -> thread
layers

- **Layer 1 – Physical**
  - Bits onto the wire
- **Layer 2 – Local link**
  - Frames to the next hop (local address)
- **Layer 3 – Internet**
  - Packets across the net to the remote machine
- **Layer 4 – Transport**
  - Payload to the target app
null
“block” parsers

- Cast “struct ip_hdr” over bytes
- Backtracking
“streaming” processors

• Byte at a time
Parsing the bytes

If you are using ntohs(), you are doing it wrong

```c
short port = ntohs(*(short*)p);
int port = p[0] << 8 | p[1];
```
GET /index.html HTTP/1.0
Host: www.example.com

parser
state
machine
void http_parse(int *state, unsigned char c, ...)
{
    switch (*state) {
    case START: ...
    case METHOD: ...
    case URI: ...
    case VERSION: ...
    case VALUE: ...
    case NAME: ...
    case COLON: ...
    case SPACE1: ...
    case SPACE2: ...
    case SPACE3: ...
    case EOL: ...
    }
}

void http_parse(int *state, unsigned char c, ...) {
    switch (*state) {
        ...
        case METHOD: /*GET, POST, HEAD, ...*/
            if (c == '\n') {
                *state = EOL;
            } else if (strchr(WHITESPACE, c)) {
                *state = SPACE1;
            } else {
                ; /*no transition*/
            }
        ...
    }
}
alert tcp any any -> any 80 (content:"GET"; http_method;)

/={h, s}

0

1 h e {he} r

2 i

3 s h

4 e

5 {he, she}

6 s

7 {his}

8 s

9 {hers}
void http_parse(int *state, unsigned char c, int *ac, ...) {
    switch (*state) {
        ...
        case METHOD:
            if (c == '\n') {
                search_end(xxMethods, ac);
                *state = EOL;
            } else if (strchr(WHITESPACE, c)) {
                search_end(xxMethods, ac);
                *state = SPACE1;
                search_start(xxSpaces, ac, c);
            } else {
                search_continue(xxMethods, ac, c);
                if (length++ > 128) ...
            }
        }
    }
    ...
}
IDS and parser state machines

• “Do you reassemble TCP?”
• “Do you swap bytes?”
• “Do you handle data normalization?”
Properties

• Low memory
  – We don’t buffer the HTTP method, we match the pattern as bytes arrive.
  – Don’t need to reassemble TCP

• Robust
  – Can’t have a buffer overflow in code that doesn’t buffer things
  – All sorts of edge cases disappear
  – Gracefully “falls off the end” of a packet
  – Don’t need to free stuff
Who does this?

• Most everyone in closed-source
  – IDS: Proventia, Intruvert, Palo Alto
  – Web: IIS
  – ... and so much more

• Some open source
  • IDS: Snort, Suricata
  – ferret.googlecode.com
    • (this sucks BTW)
DFA Performance

- state = table[state][c];
- mov ebx[eax+ecx],%eax
One L1 cache hit per byte

- mov ebx[eax+ecx],%eax
  mov ebx[eax+ecx],%eax
  mov ebx[eax+ecx],%eax
  mov ebx[eax+ecx],%eax
- 3 GHz CPU with 3 Hz cache – 8-gbps
switch/case

- 15 Hz branch misprediction
- 0 Hz branch prediction
Special case

- Theory: byte-at-a-time
- Practice: grab multiple bytes that are part of the same state
[http-parse]

• https://github.com/robertdavidgraham/papers/blob/master/state-machine-perf/http-parse.c
[masscan]

https://github.com/robertdavidgraham/masscan
• SSL state machine parser:
  - https://github.com/robertdavidgraham/masscan/blob/master/src/proto-ssl.c

• X.509 certificate parser:
  - https://github.com/robertdavidgraham/masscan/blob/master/src/proto-x509.c