A load balancer :
1- receives requests/connections from the client
2- selects a server (“takes a routing decision”)
3- forwards requests/connections to the server
4- receives the response from the server
5- forwards the response to the client
6- may goto 1 if needed (eg: persistent conns)
7- tries to do this as quickly as possible

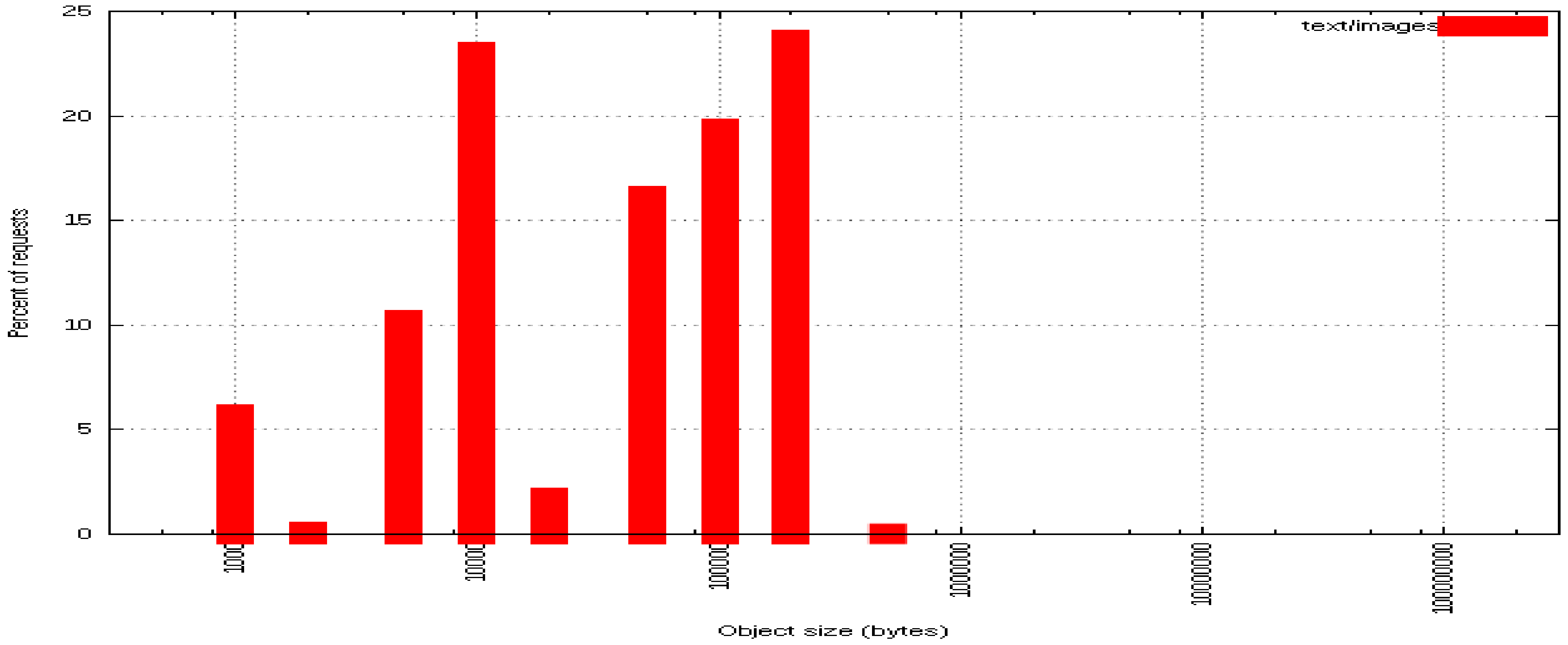
General implementation :
- moderately layered approach
- try to make the lower parts do most of the work
- critically relies on operating system (70 to 99%)

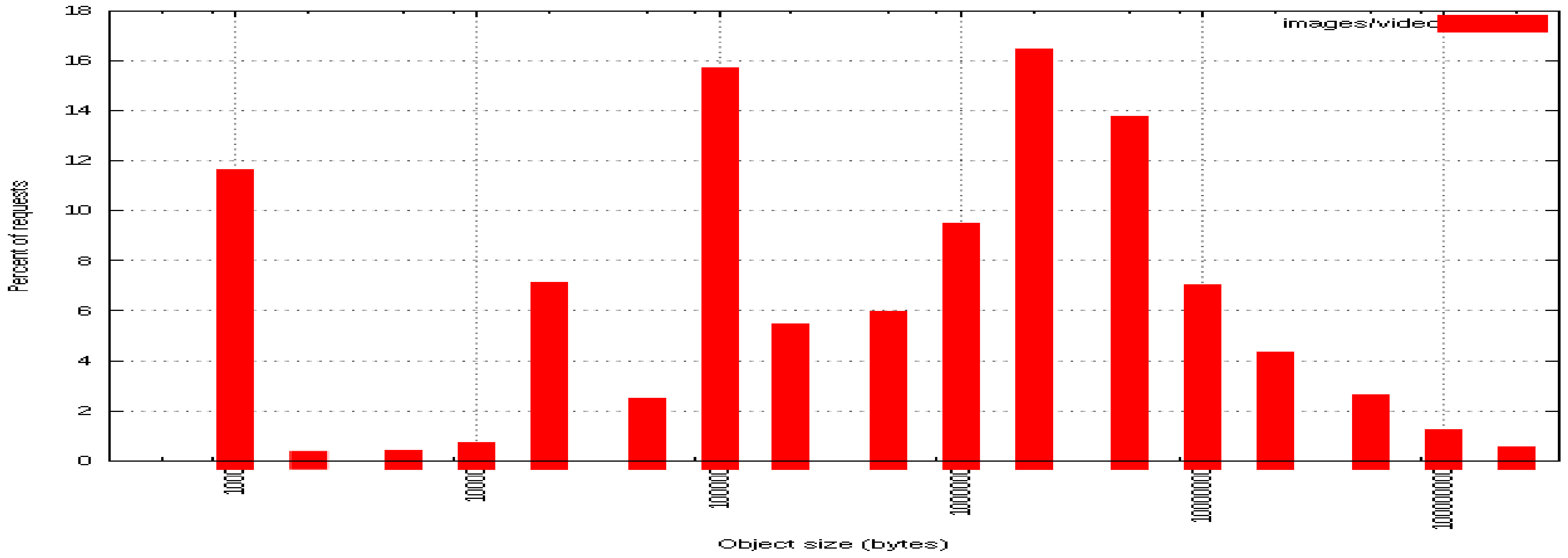
Challenges :
- find the routing key for the initial request
  (host header, URL, cookie)
- find the routing key for subsequent requests
  => needs to parse framing after routing

Layered analysis of a real world web site made of two sub-sites :
- 1 composed of HTML + images, avg 52 kB (3% of the traffic in bytes)
- 1 composed of videos and redirects, avg 3.7 MB (97% of the traffic in bytes)

Points of consideration :
- important traffic, a few 100s of Gbps, tens of LBs
- currently HTTP/1, studies migration to HTTPS, and ultimately HTTP/2
=> goal: estimate #of LBs and CPU cores needed

Note: **such users are willing to significantly adjust their architecture if needed**.


text/images
Percent of requests
Object size (bytes)
0
5
10
15
20
25
1000
10000
100000
1000000
10000000
100000000

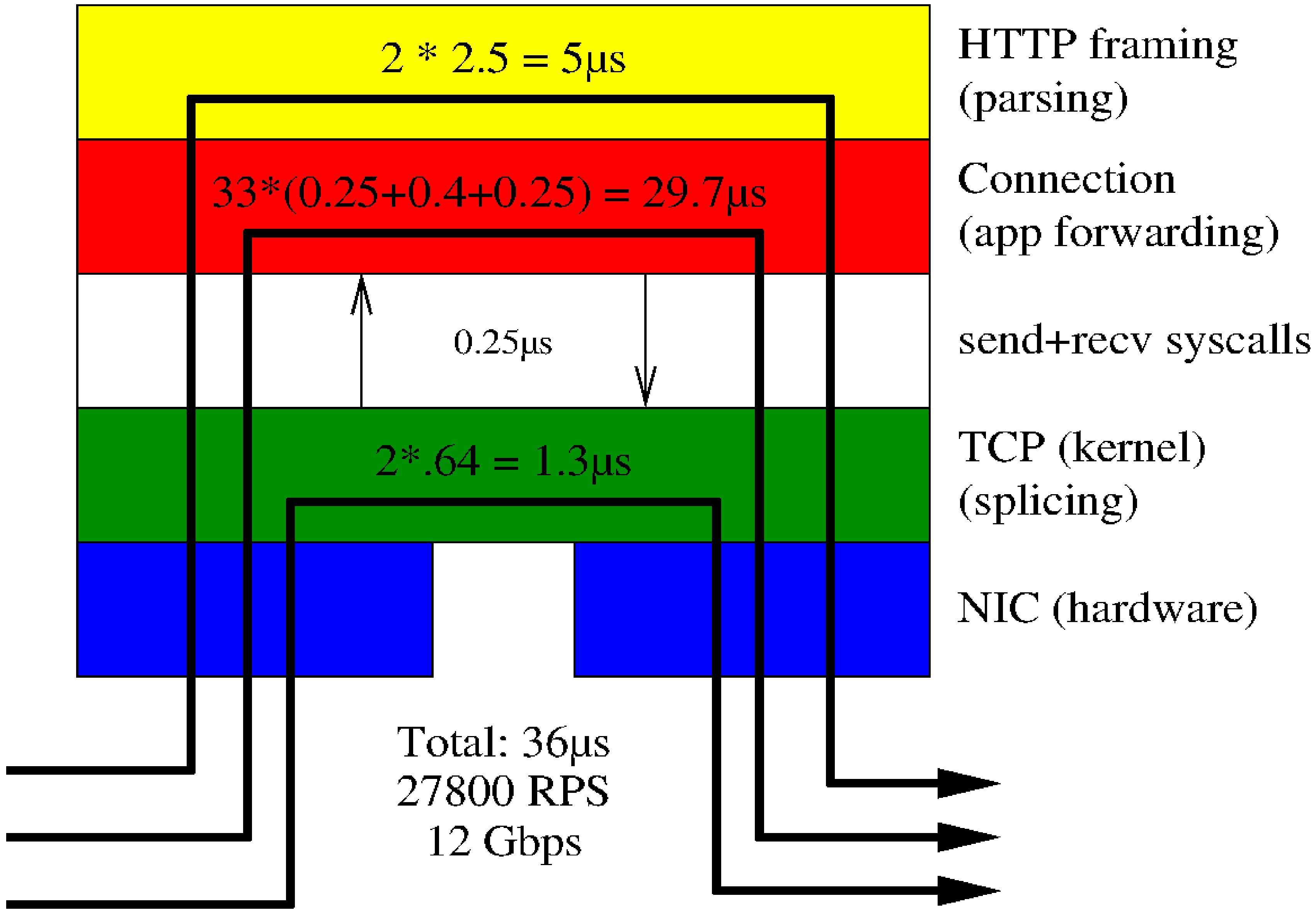

images/video
Percent of requests
Object size (bytes)
0
2
4
6
8
10
12
14
16
18
1000
10000
100000
1000000
10000000
100000000

Tests were run on a Xeon E5-1630v3 (4x3.7 GHz) with a dual-port intel 10G NIC, haproxy-1.6, openssl-1.0.2 + AES-NI, for object sizes of 52 kB and 3.7 MB.

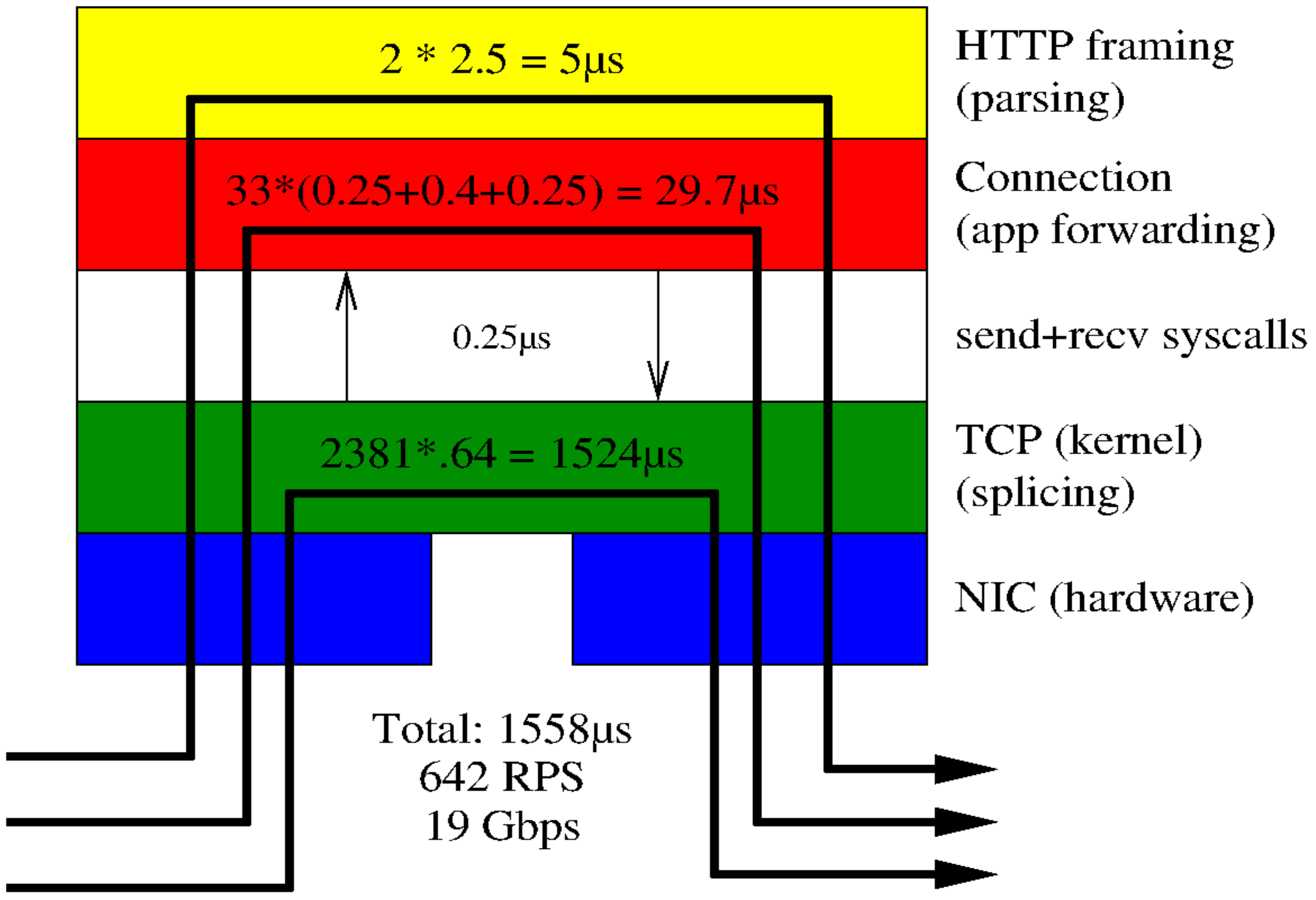
The approximative processing time per TCP segment (1460) for each layer is measured and reported.

Request and response processing count as one TCP segment each.

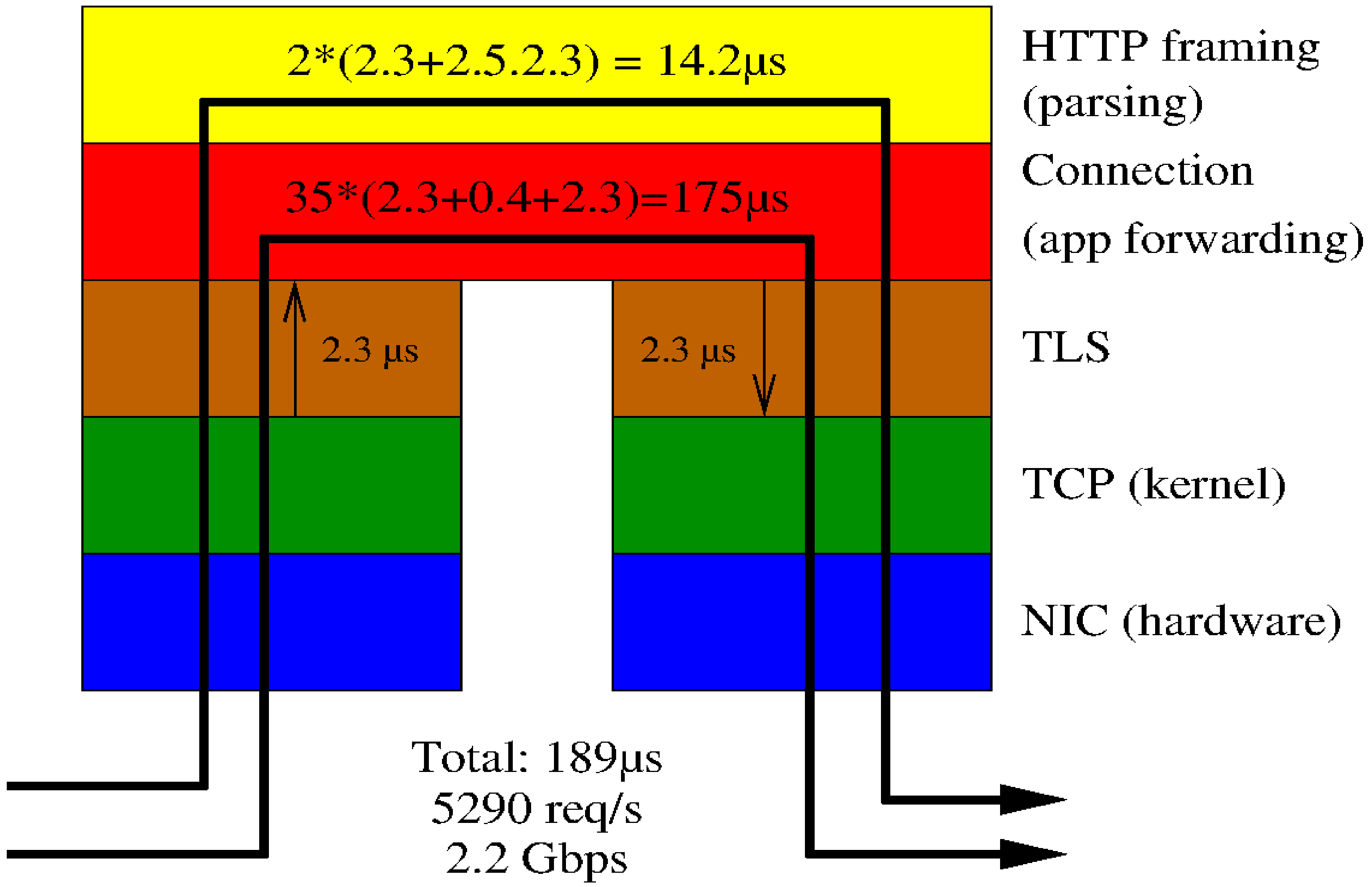
HTTP/1 clear : 52kB object (1+36 segments) : 12 Gbps/core



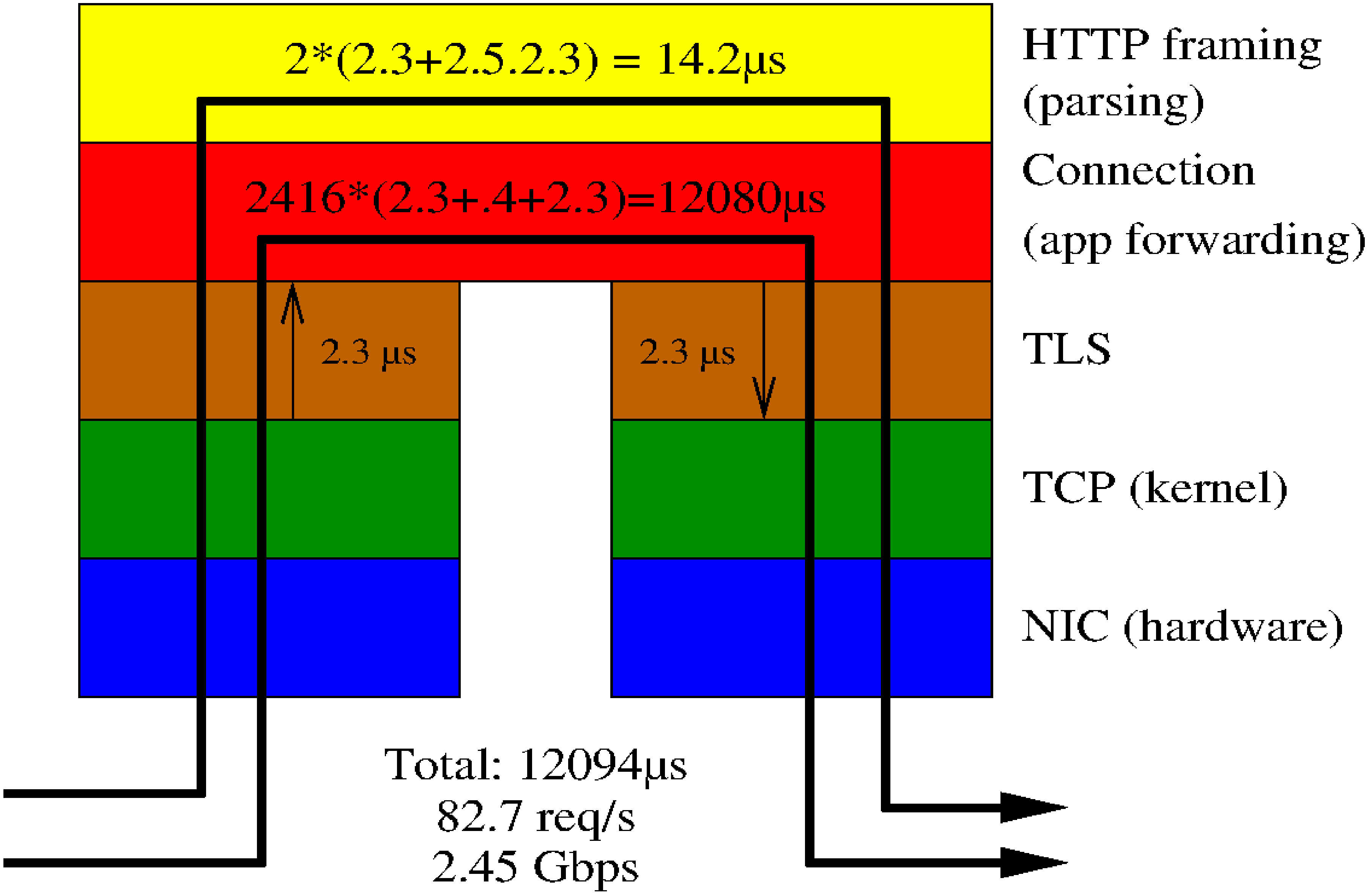
HTTP/1 clear : 3.7MB object (1+2417 segments) : 19 Gbps/core



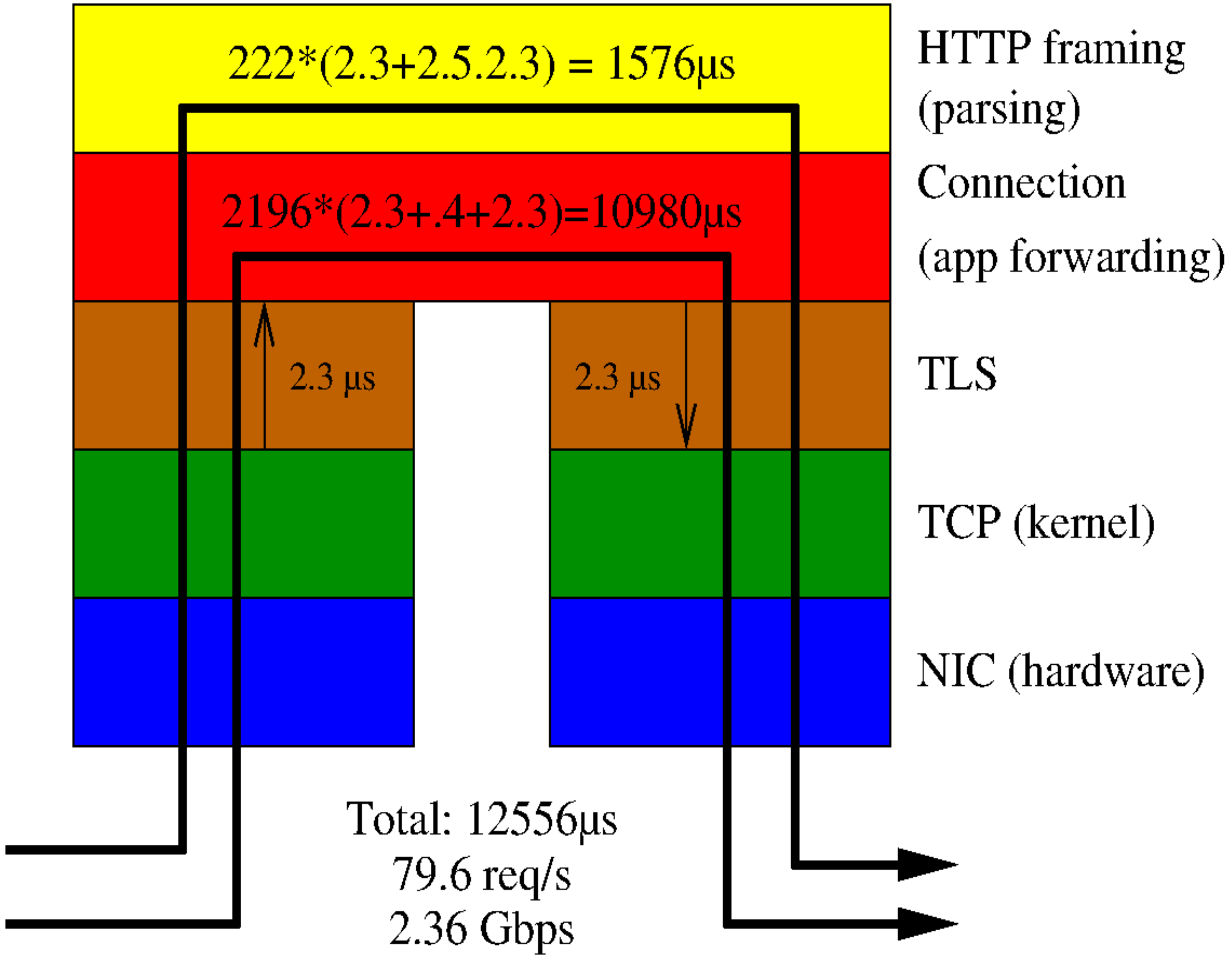
HTTP/1+TLS : 52kB object 1+36 segments) : 2.2 Gbps/core



HTTP/1+TLS :3.7MB object (1+2417 segments) : 2.45 Gbps/core



HTTP/2 default frame size:3.7MB object (1+2417 segments) : 2.36 Gbps/core



First observations :
- HTTPS on both sides will require **8** cores per **20** Gbps instead of **1** core per **20** Gbps
- only 4 cores per 20 Gbps by not encrypting to the server
- key generation not accounted for (first site sees 2400 new visitors per second)

=> **But why do we have to decrypt in the first place ?**

Can we do better ?
- host header can be extracted from SNI without decrypting.
- URL can be ignored in properly architected sites, unless URL hashing is desired (cache deduplication)
- cookies are only used to store/retrieve the last server that processed the request

Simple proposal :
Add two TLS extensions to transport the strict minimum needed :
- a **32-bit hash** (or less) of the URL performed by the client
- a **16-bit server identifier** sent by the LB to the client and repeated by the client on subsequent connections

=> **no need to decrypt anymore**, TLS could be processed as an opaque connection at wire speed. Fallbacks on current methods when information is missing.