Software segmentation offloading for FreeBSD

Stefano Garzarella, Luigi Rizzo
stefano.garzarella@gmail.com

Università di Pisa

September 28, 2014

Outline

- Introduction
- Hardware offloading (TSO, RSC)
- Software offloading (LRO)
- Generic Segmentation Offload (GSO)
- Results

Introduction

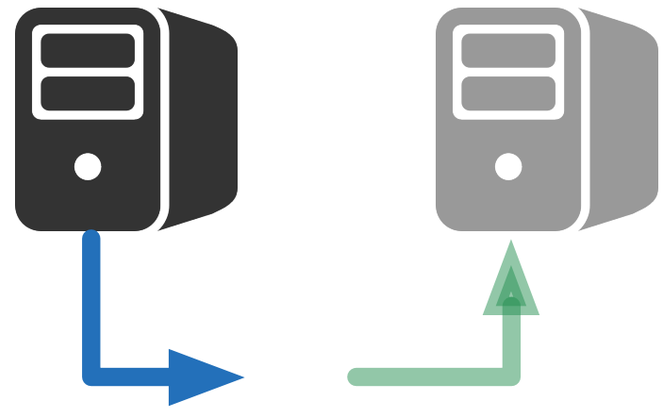
- The use of large frames makes network communication much less demanding for the CPU.

- Backward compatibility and slow links requires the use of **1500** byte or smaller frames.

- Modern NICs with hardware TCP segmentation offloading (**TSO**) address this problem.

- Generic software version (**GSO**) provided by the OS has reason to exist, for use on paths with no suitable hardware.

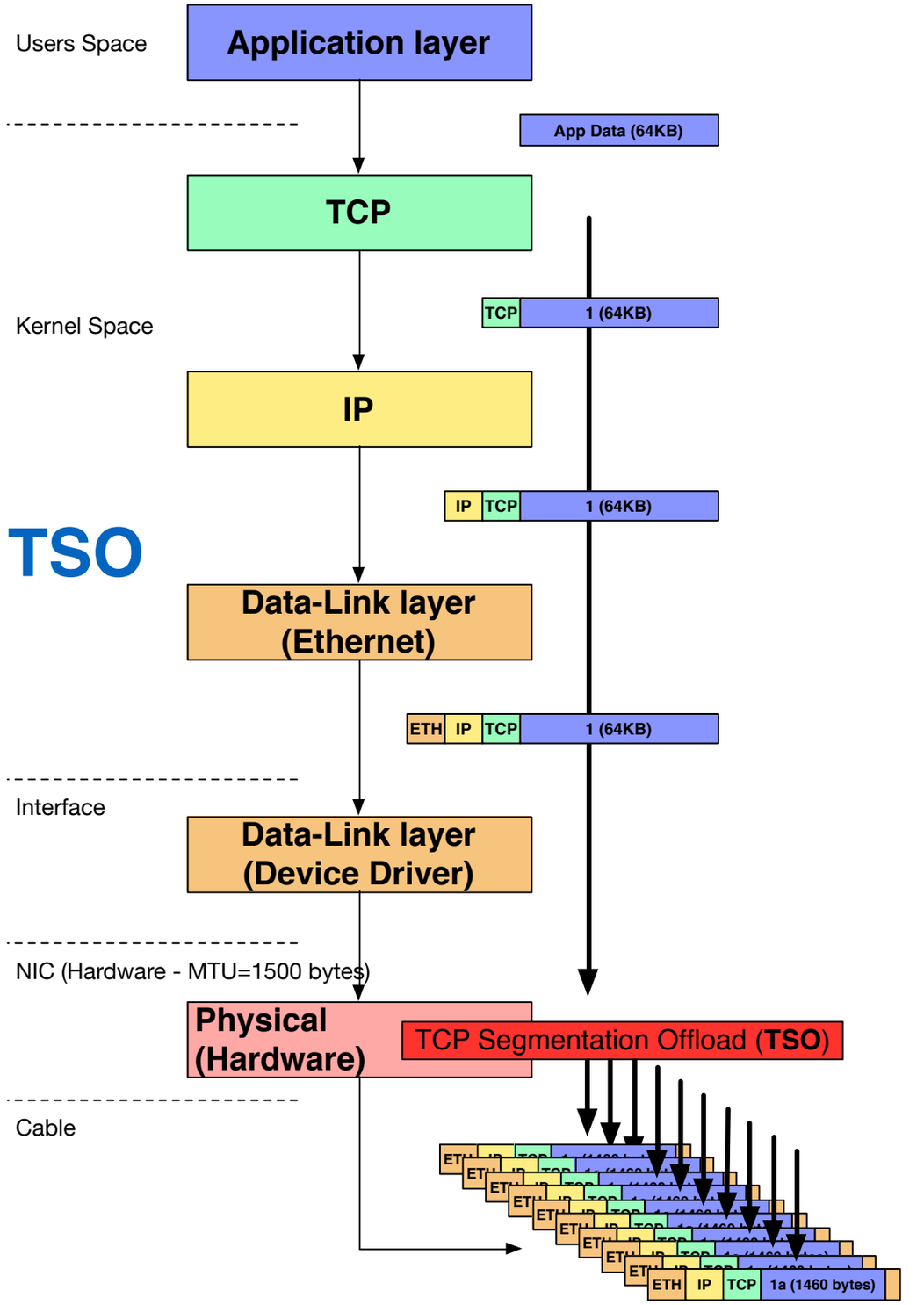
Hardware Offloading



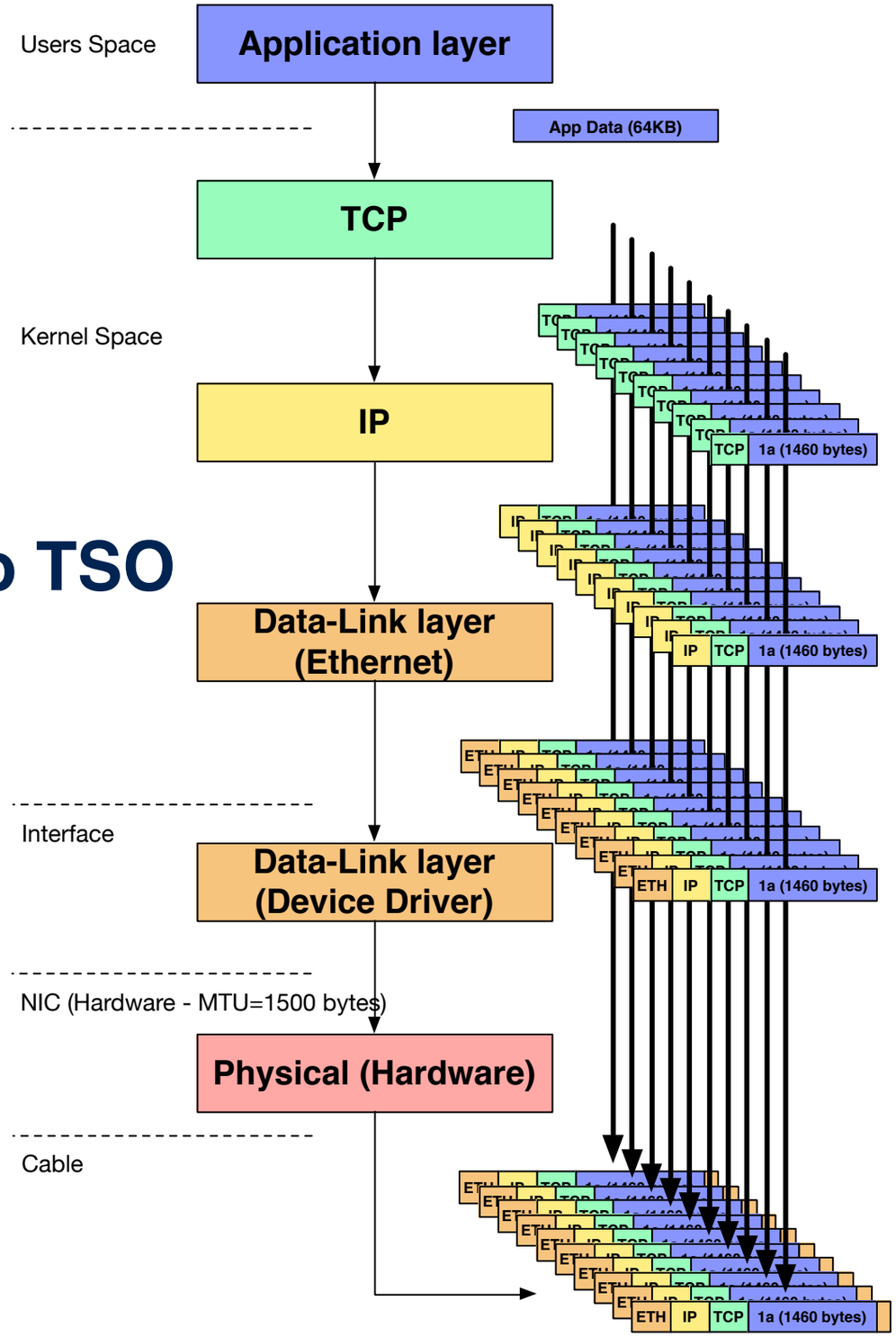
Sender side

- TCP Segmentation Offload (**TSO**)
  - Data segmentation is offloaded to the NIC, that divides the data into the maximum transmission unit (MTU) size of the outgoing interface.
  - The network stack is crossed only once per (large) segment instead of once per 1500-byte frames.
  - Issues:
    - Only TCP traffic
    - Early version supported only IPv4

TCP Segmentation Offload



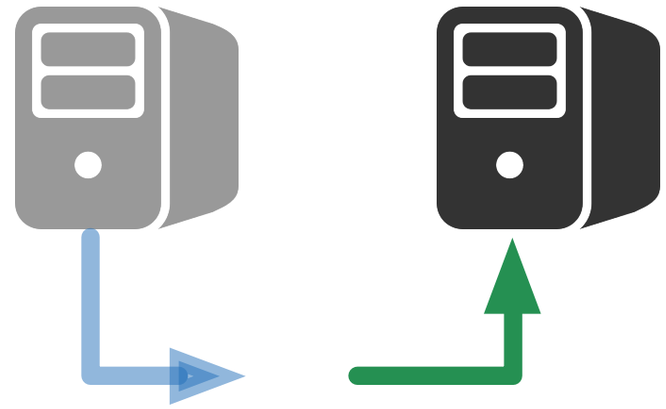
a) TSO



b) no TSO

Hardware Offloading

Receiver side



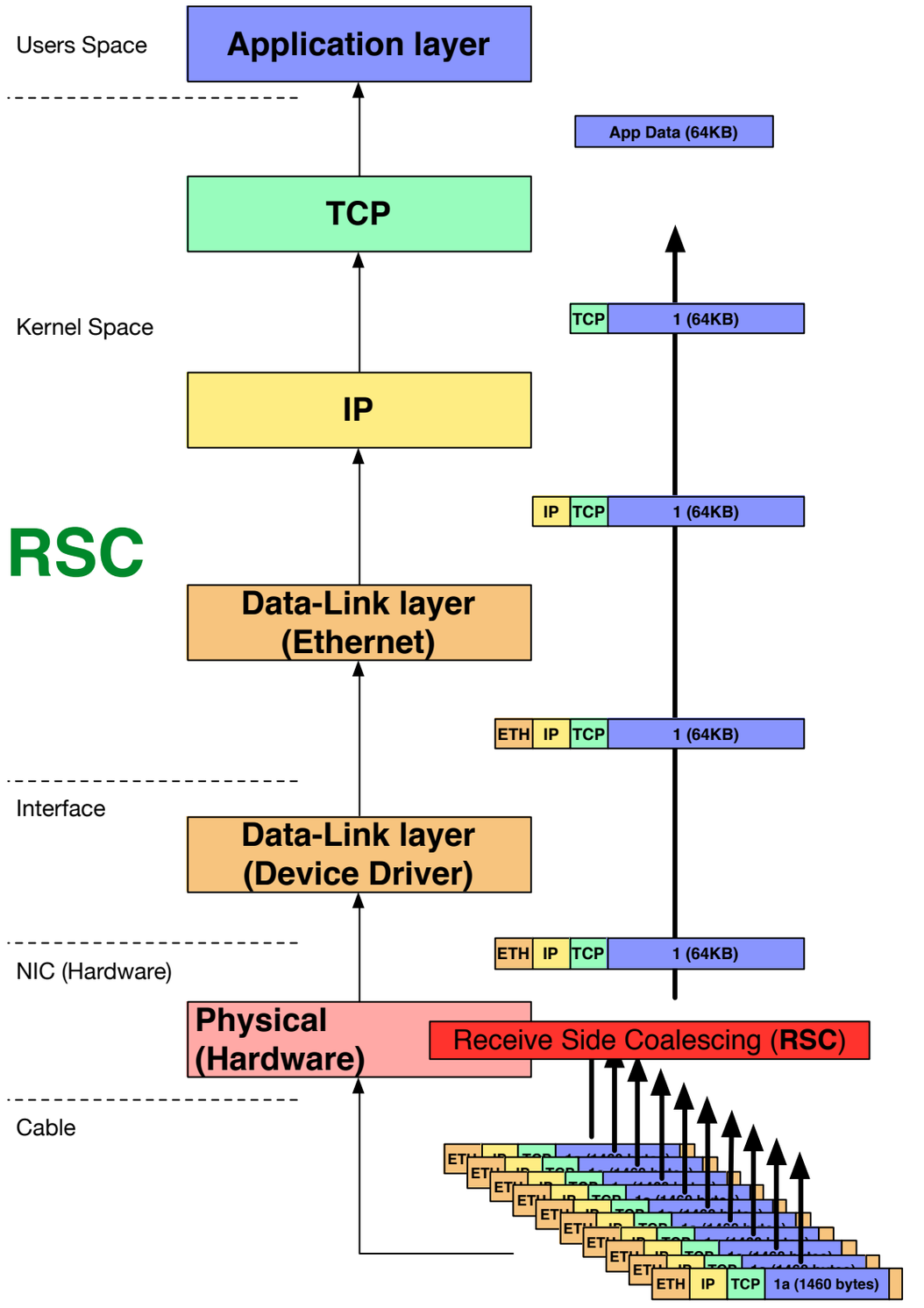
- Receive Side Coalescing (**RSC**)

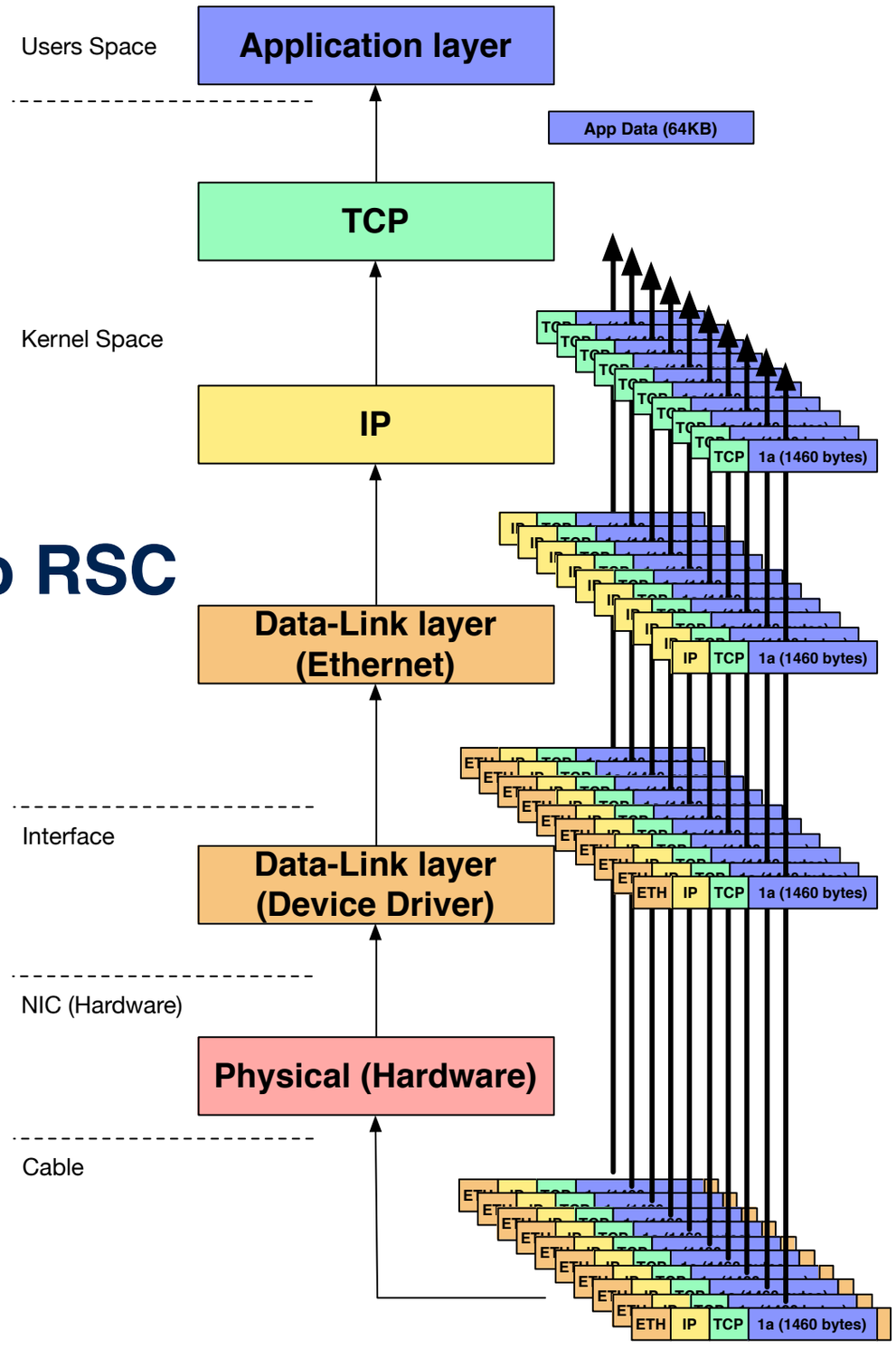
  or hardware LRO

  - Allows a NIC to combine incoming TCP/IP packets that belong to the same connection into one large receive segment before passing it to the operating system.

  - Reduces CPU use because the TCP/IP stack is executed only once for a set of received Ethernet packets.

Receive Side Coalescing

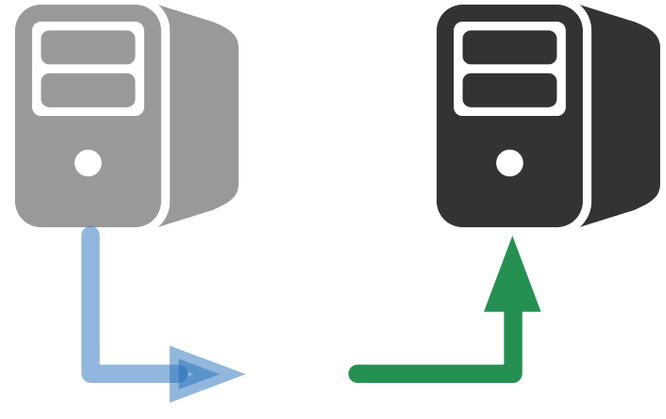




Software offloading

- Why software implementation?
  - Older NICs (IPv4 / IPv6)
  - Buggy NICs
  - Communication between Virtual Machines
  - Easy to extend for new protocols

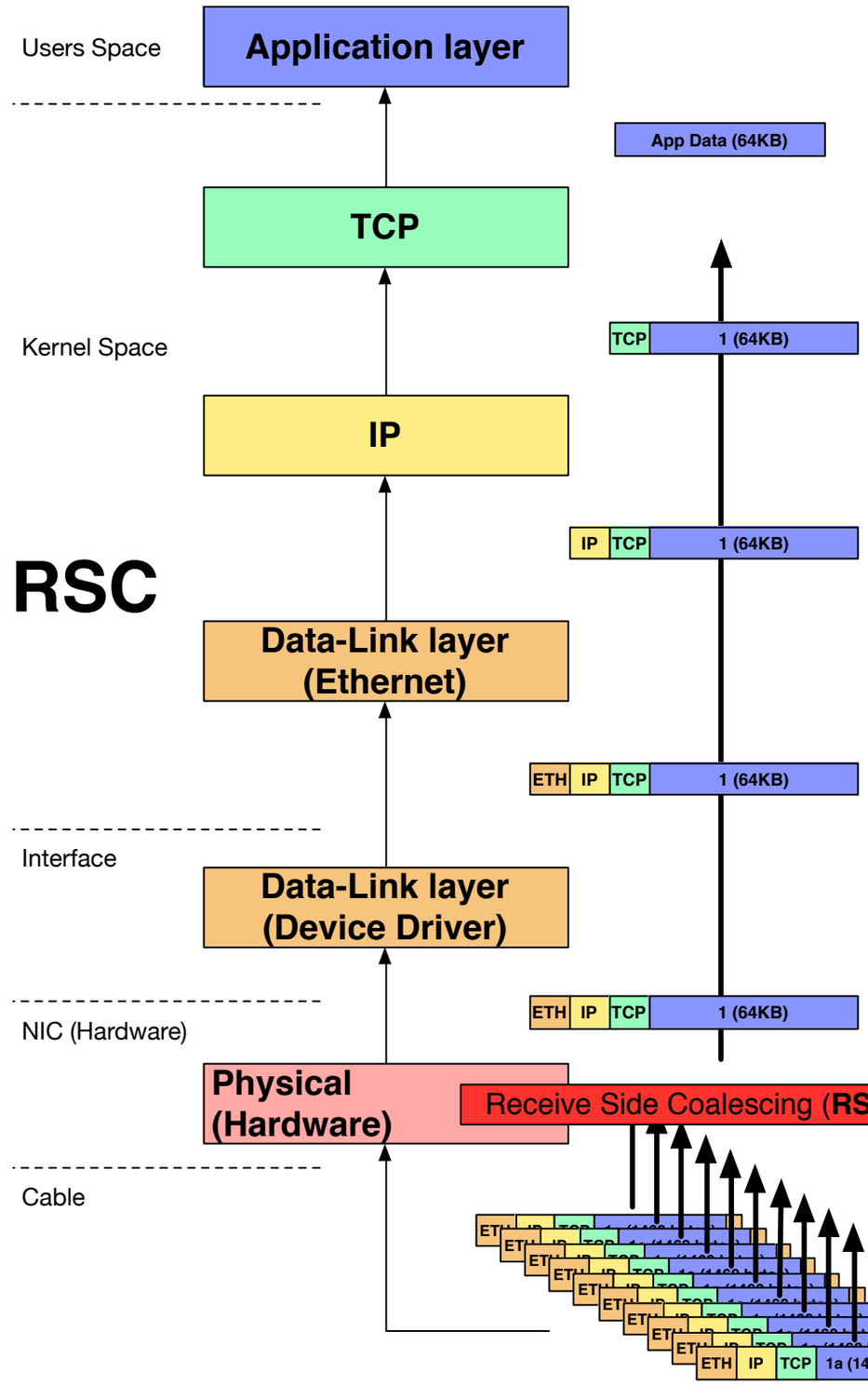
Software offloading

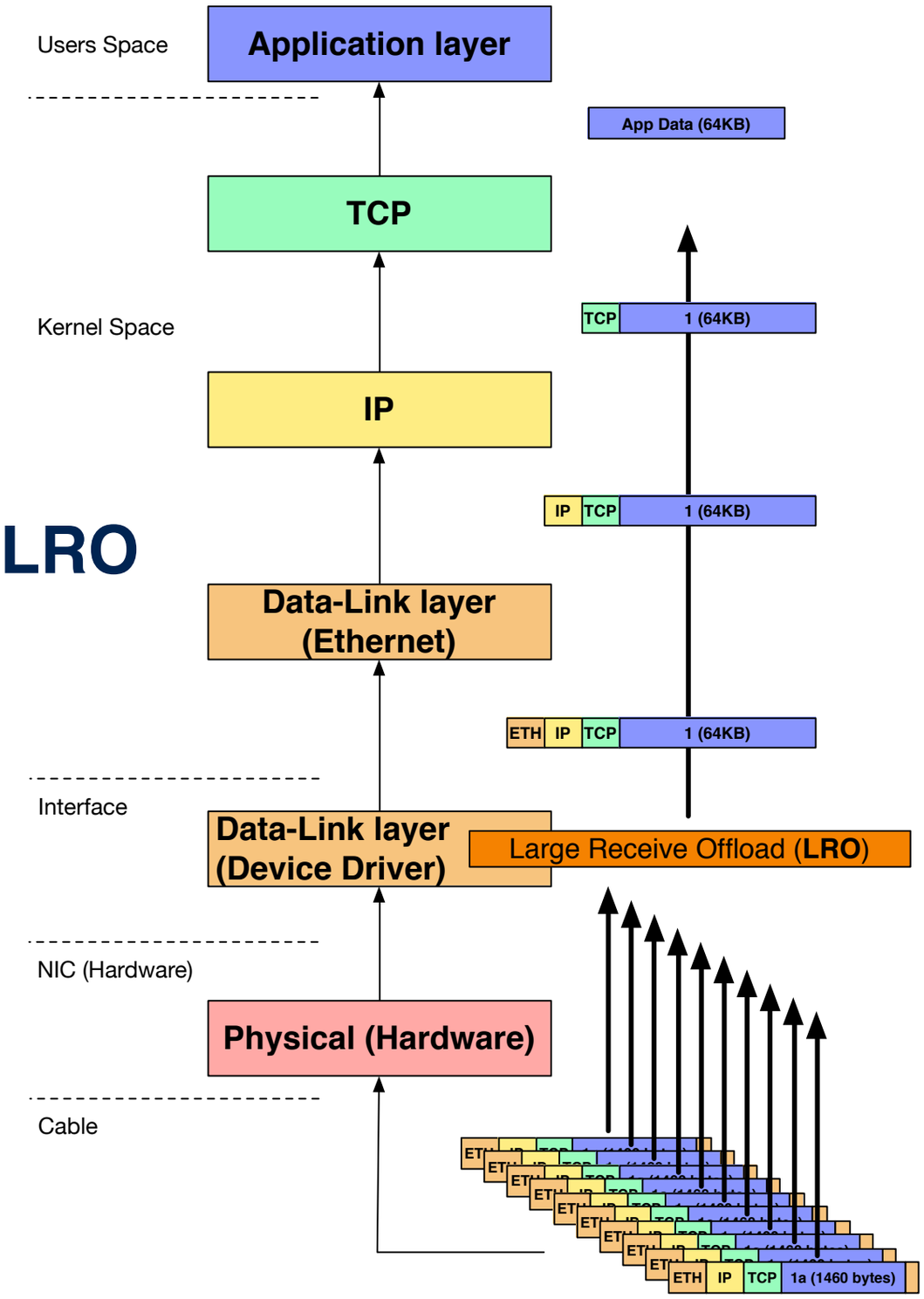


Receiver side

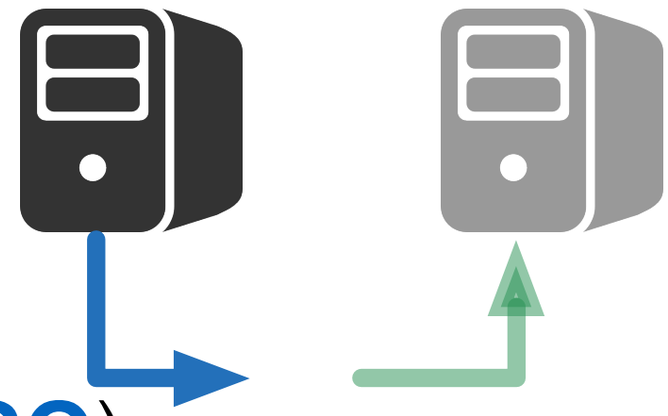
- Large Receive Offload (**LRO**)
- Software implementation of RSC
- Available since FreeBSD 7.1
- Requires changes in each device drivers

Large Receive Offload





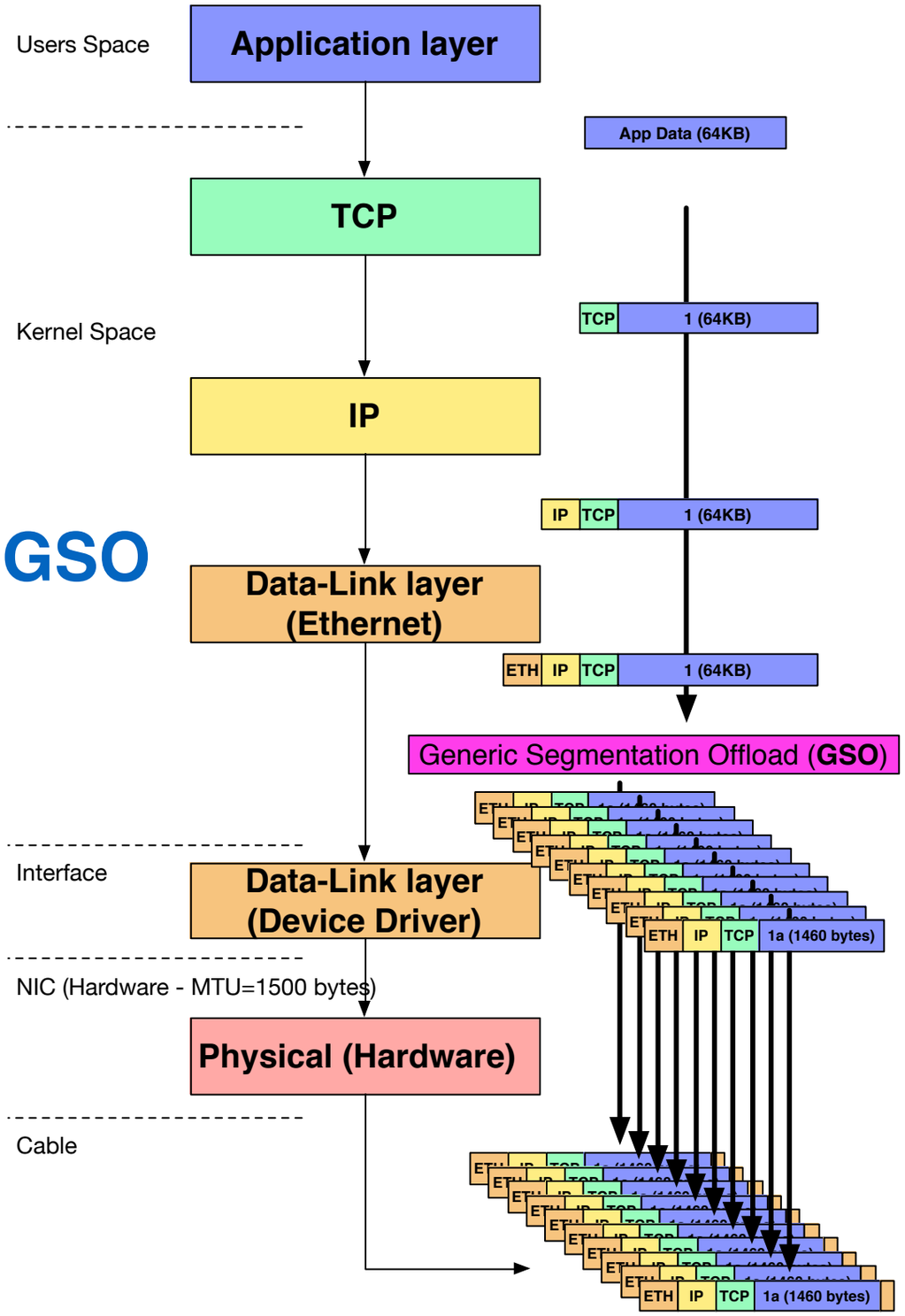
Software offloading



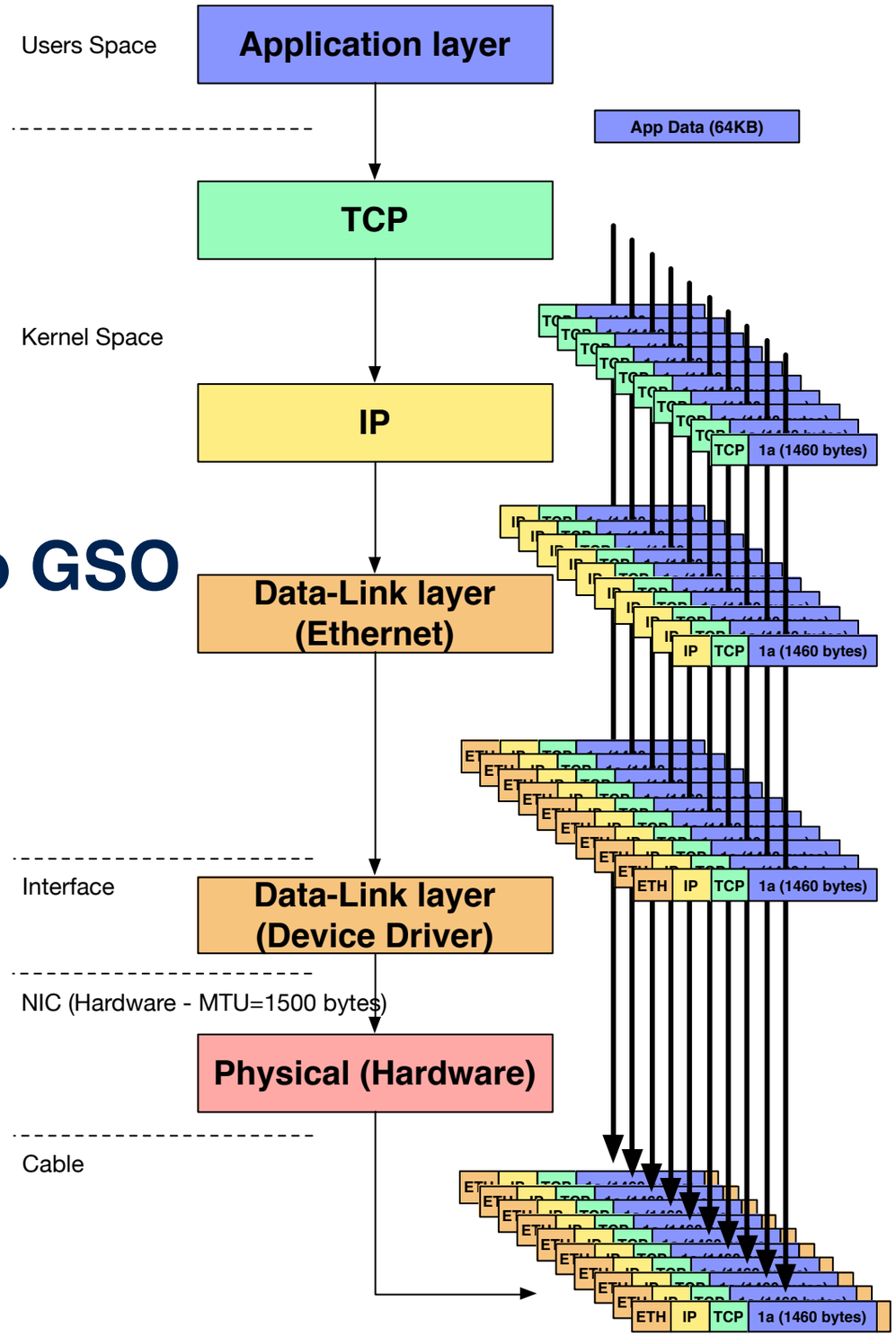
Sender side

- **Generic Segmentation Offload** (**GSO**)
  - Software implementation of TSO
  - Supported TCP, UDP and IPv4, IPv6
  - Available for FreeBSD:
    - -current, 10-stable, 9-stable
  - Segmentation just before the device driver

GSO on TCP flow



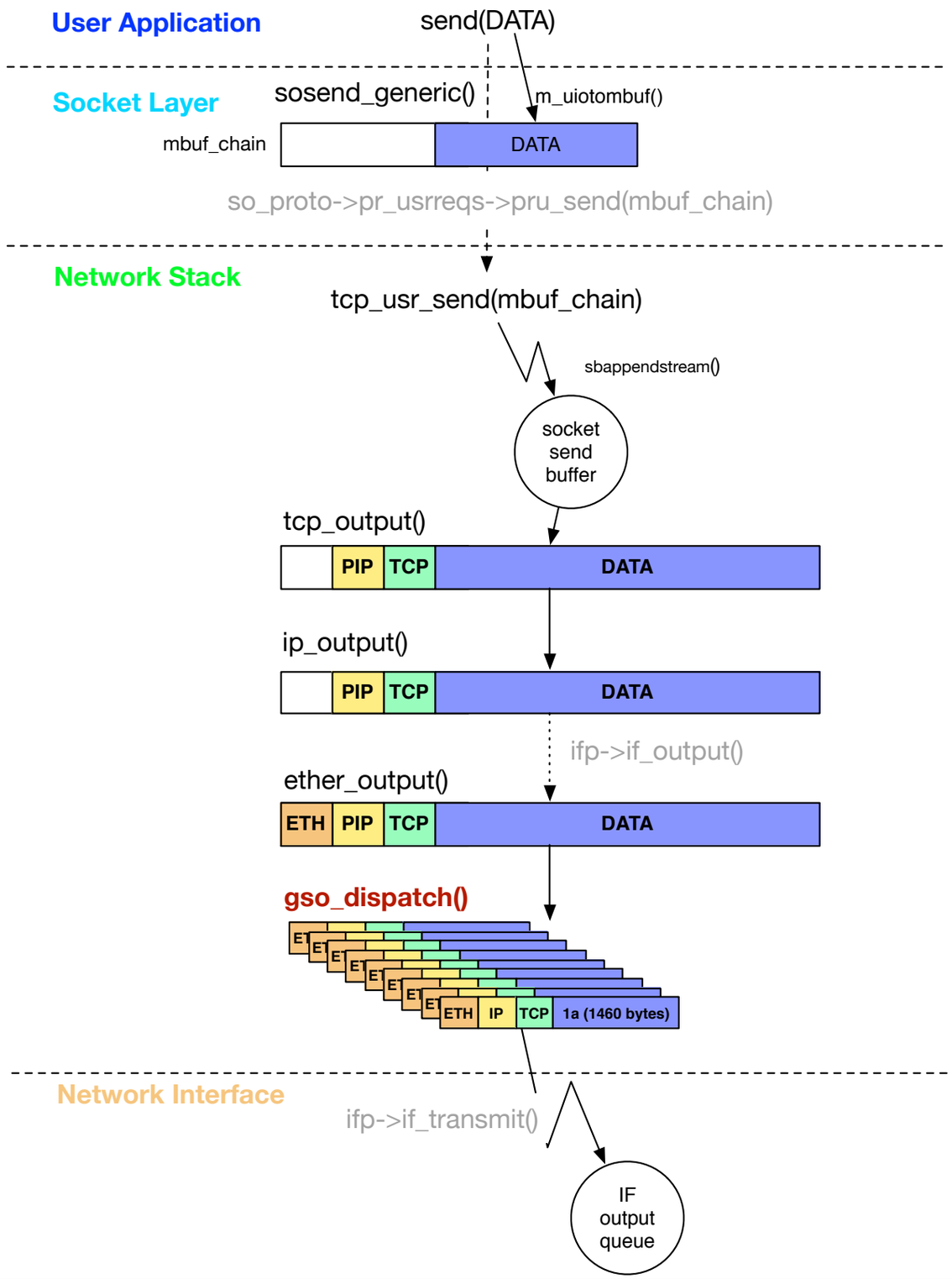
a) GSO



b) no GSO

GSO: data flow

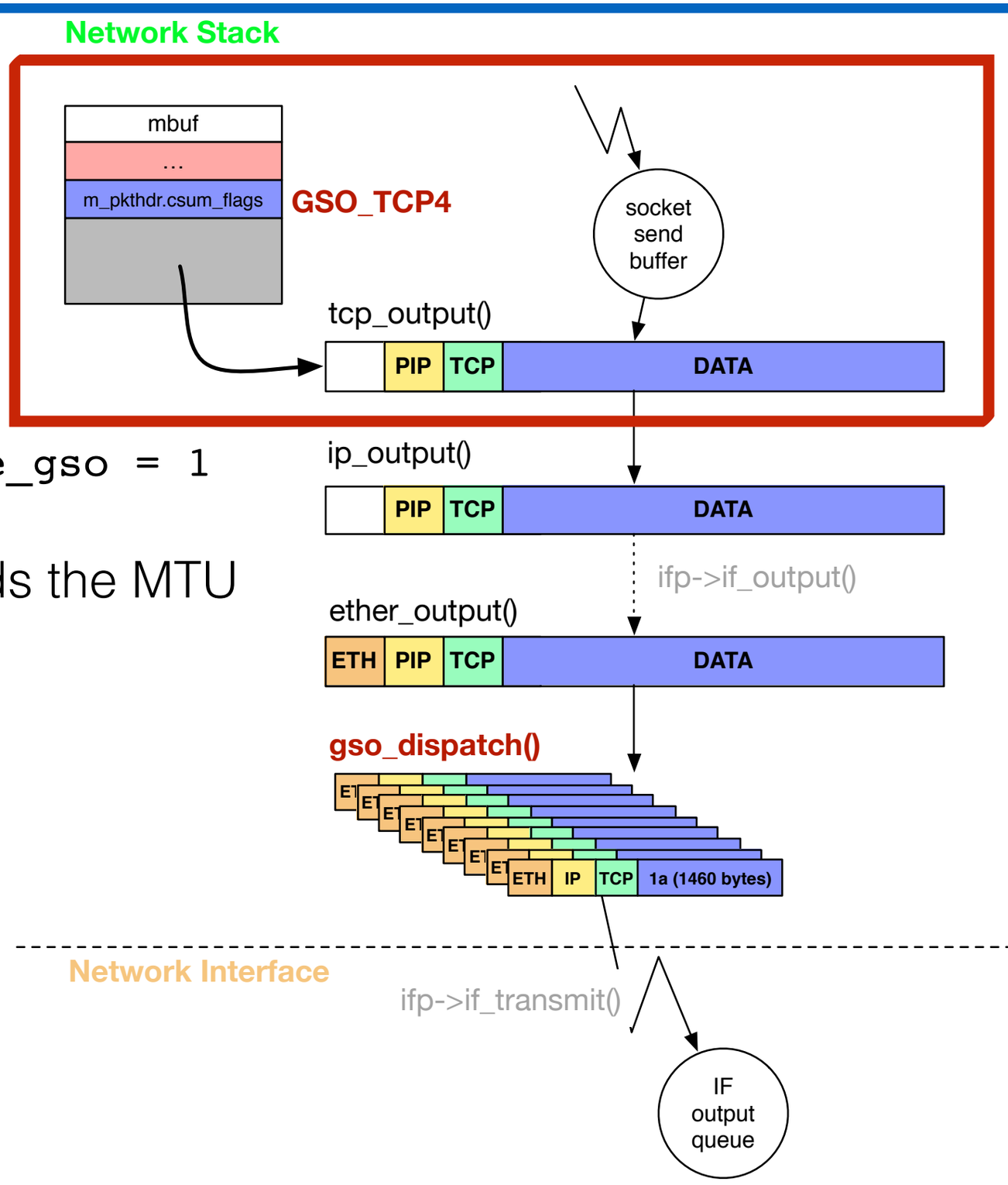
- Network stack changes
  - tcp_output()
  - ip_output()
  - ether_output()
  - gso_dispatch()



tcp_output()

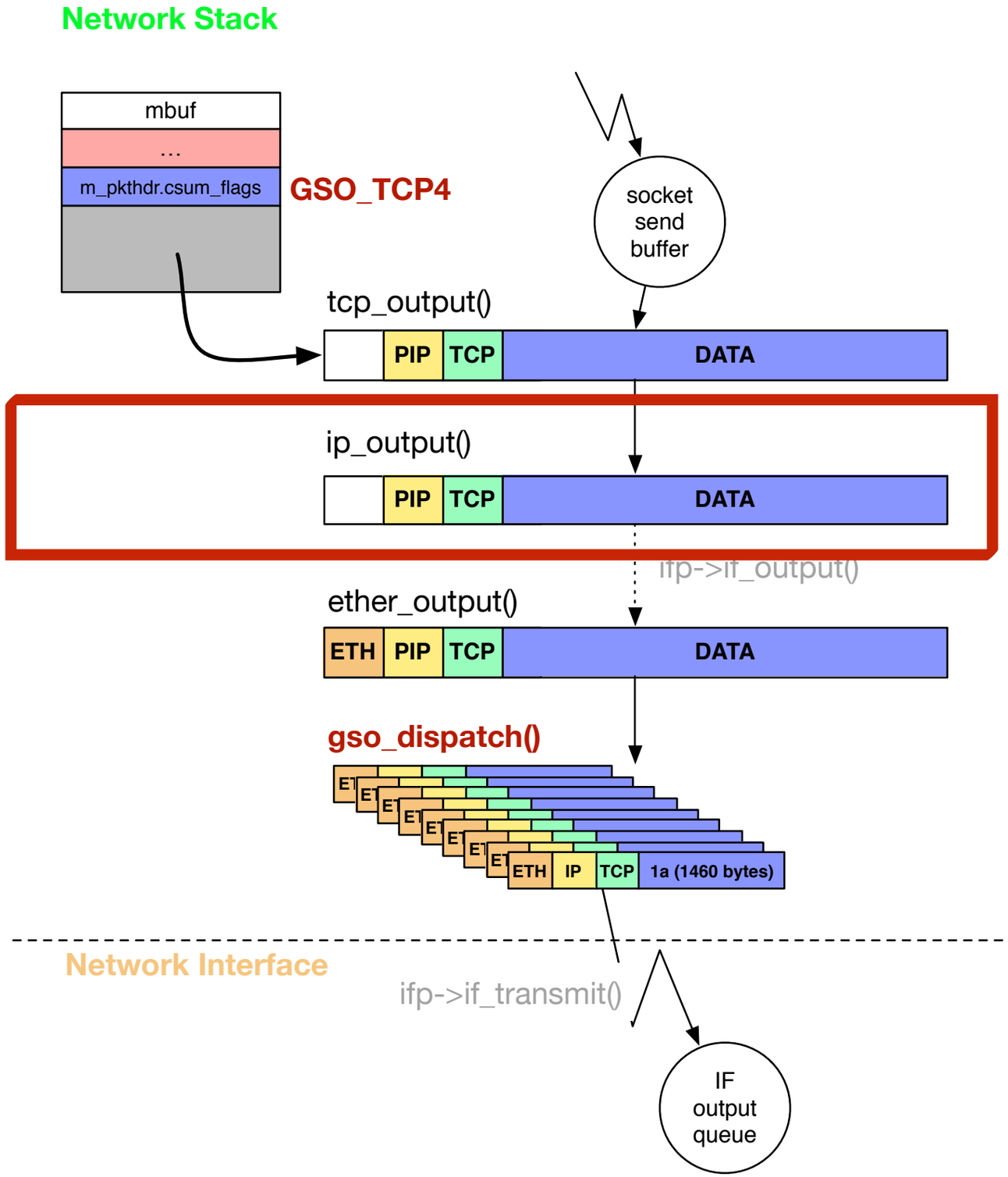
1. Checks if GSO is enabled:
   - `sysctl net.inet.tcp.gso = 1`
   - `sysctl net.gso.”ifname”.enable_gso = 1`
2. Checks if the packet length exceeds the MTU

- If 1 and 2 are true, sets GSO flag
  - `m->m_pkthdr.csum_flags |= GSO_TO_CSUM(GSO_TCP4);`



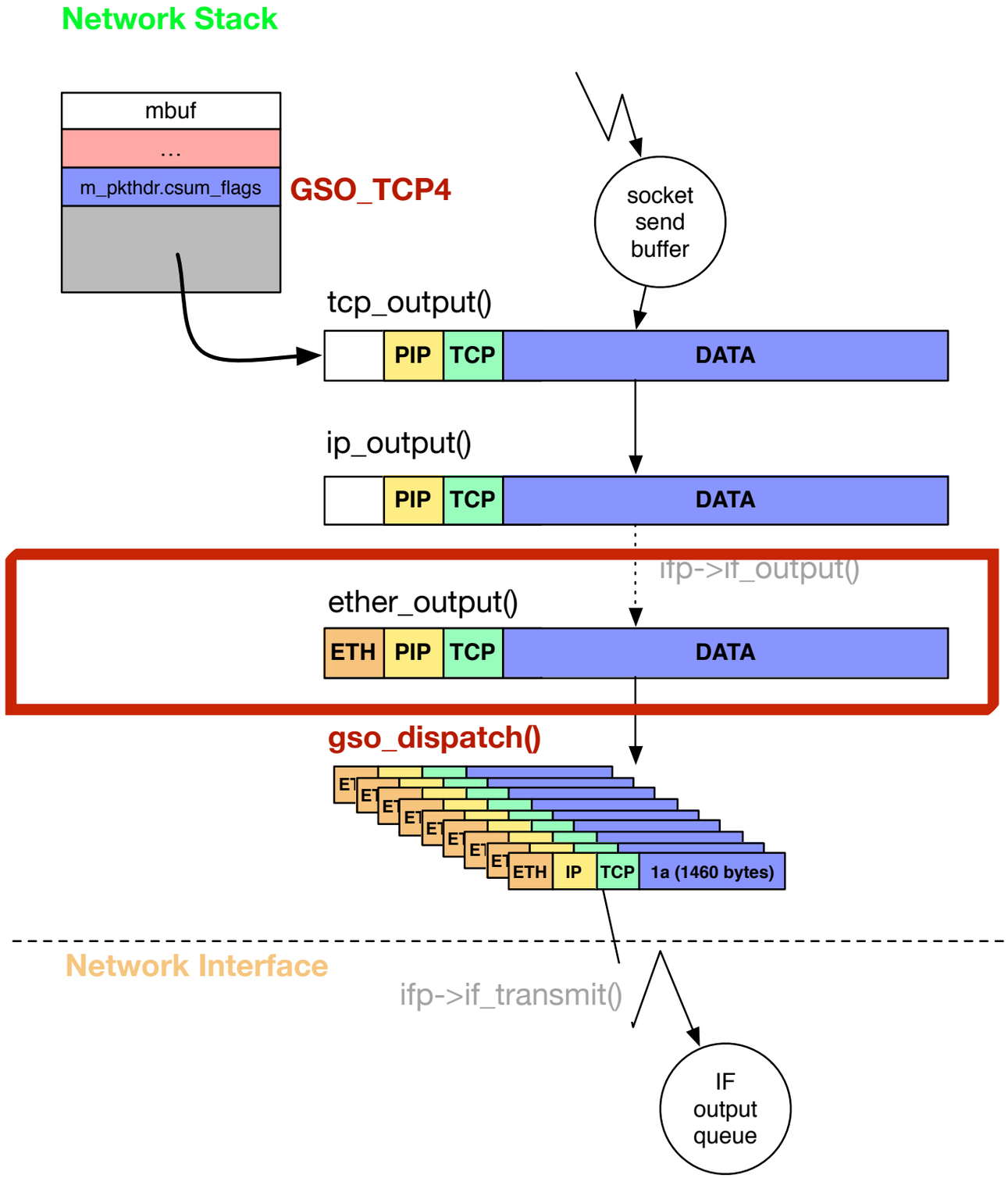
ip_output()

- If GSO is enabled and required:
- avoids checksum (IP & TCP)
- avoids IP Fragmentation



ether_output()

- If GSO is enabled and required:
  - calls `gso_dispatch()` instead of `ifp->transmit()`

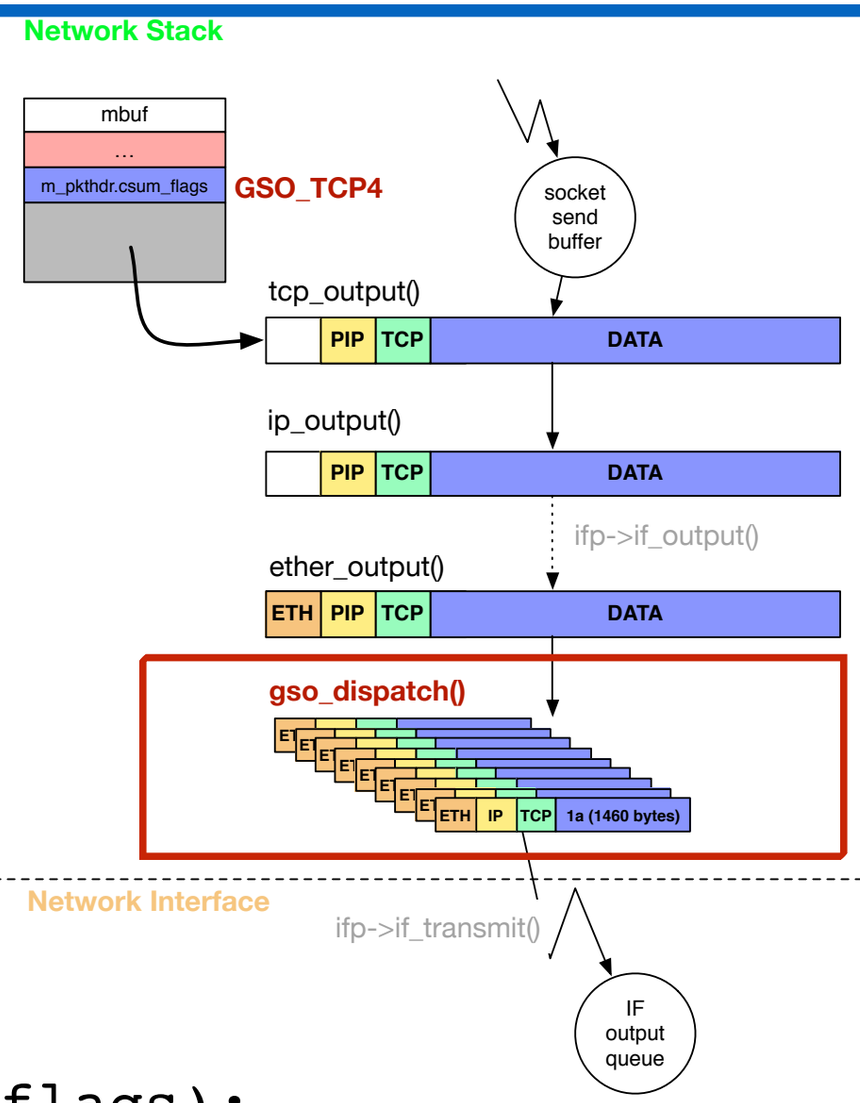


gso_dispatch()

```
enum gso_type {
     GSO_NONE,
     GSO_TCP4,
     GSO_TCP6,
     GSO_UDP4,
     GSO_UDP6,
 /*
  *  GSO_SCTP4, TODO
  *  GSO_SCTP6,
  */
     GSO_END_OF_TYPE
 };
```

```
int (*gso_functions[GSO_END_OF_TYPE])
            (struct ifnet*, struct mbuf*, u_int);

#define CSUM_TO_GSO(x) ((x & CSUM_GSO_MASK)    \
                                >> CSUM_GSO_OFFSET)
```




```
int
gso_dispatch(struct ifnet *ifp,
             struct mbuf *m, u_int mac_hlen)
{
    …
    gso_flags = CSUM_TO_GSO(m->m_pkthdr.csum_flags);
    …

    error = gso_functions[gso_flags](ifp, m, mac_hlen);
    return error;
}
```

gso_functions[]

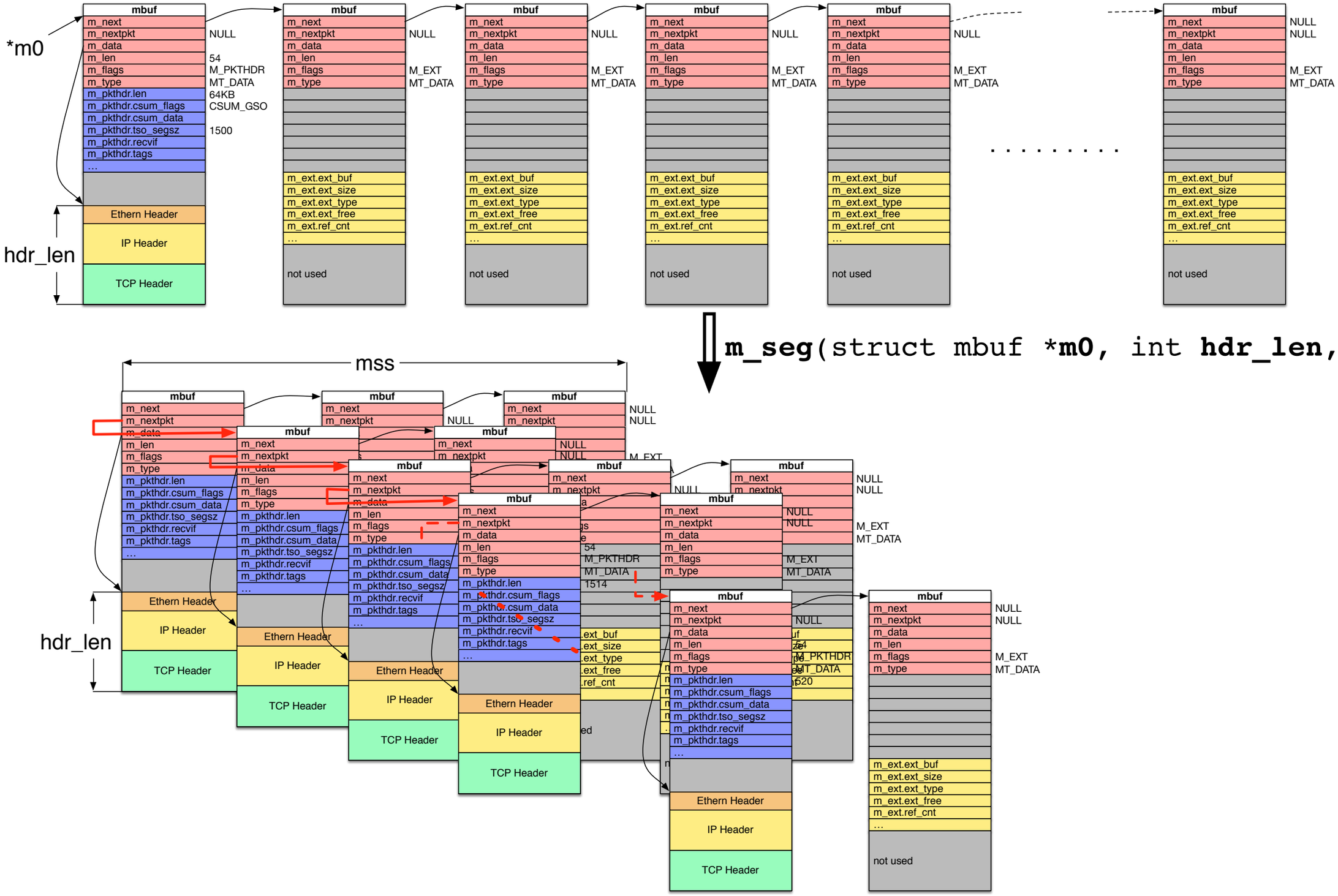
- `gso_functions[GSO_TCP4]`
  - `gso_ip4_tcp(…)` - GSO on TCP/IPv4 packet

    1. `m_seg(struct mbuf *m0, int hdr_len, int mss, …)`
       returns the mbuf queue that contains the segments of the original packet (m0).
       - hdr_len - first bytes of m0 that are copied in each new segments
       - mss - maximum segment size

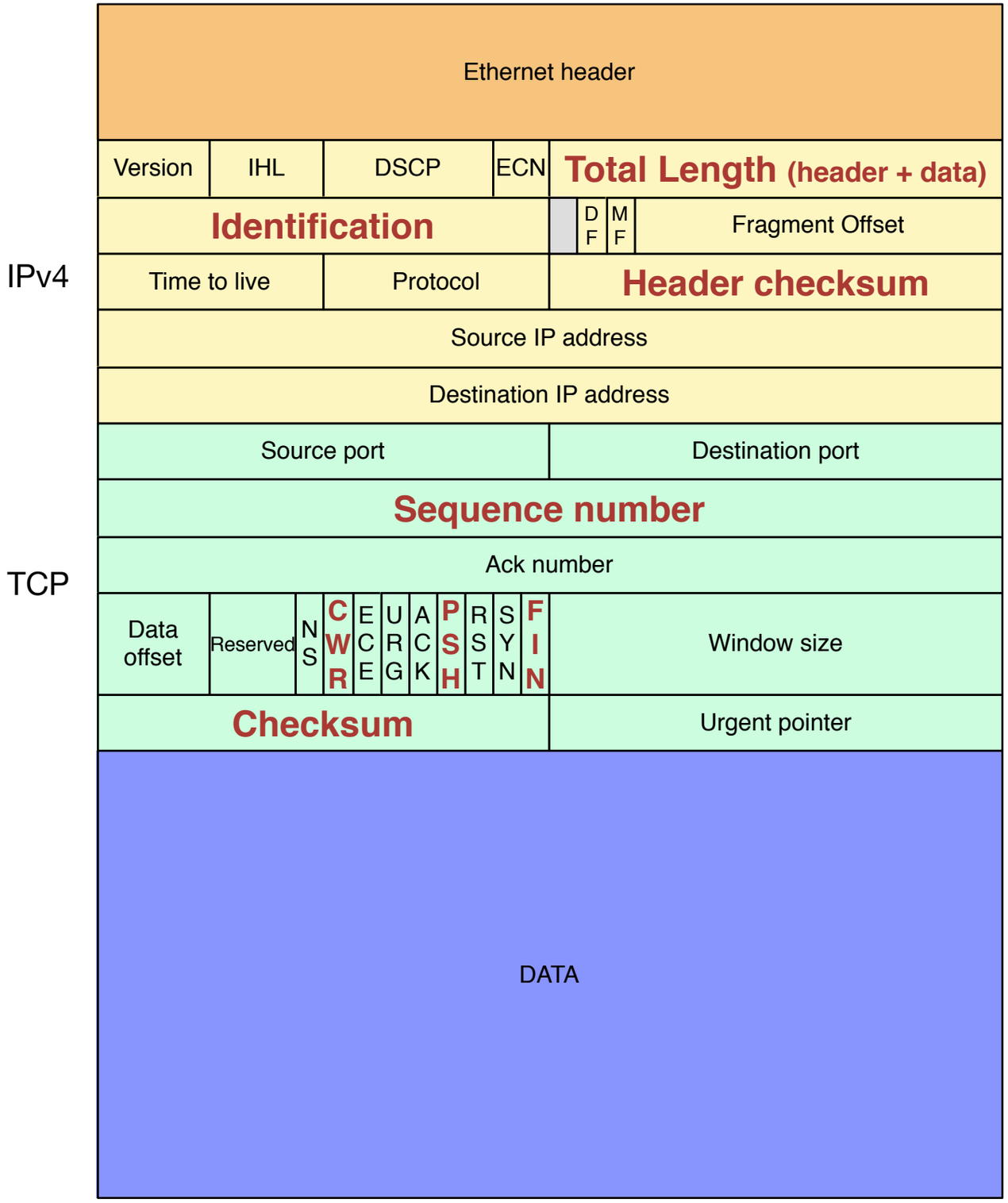
    2. fixes TCP and IP headers in each new segments

    3. sends new segments to the device driver [`ifp->if_transmit()`]

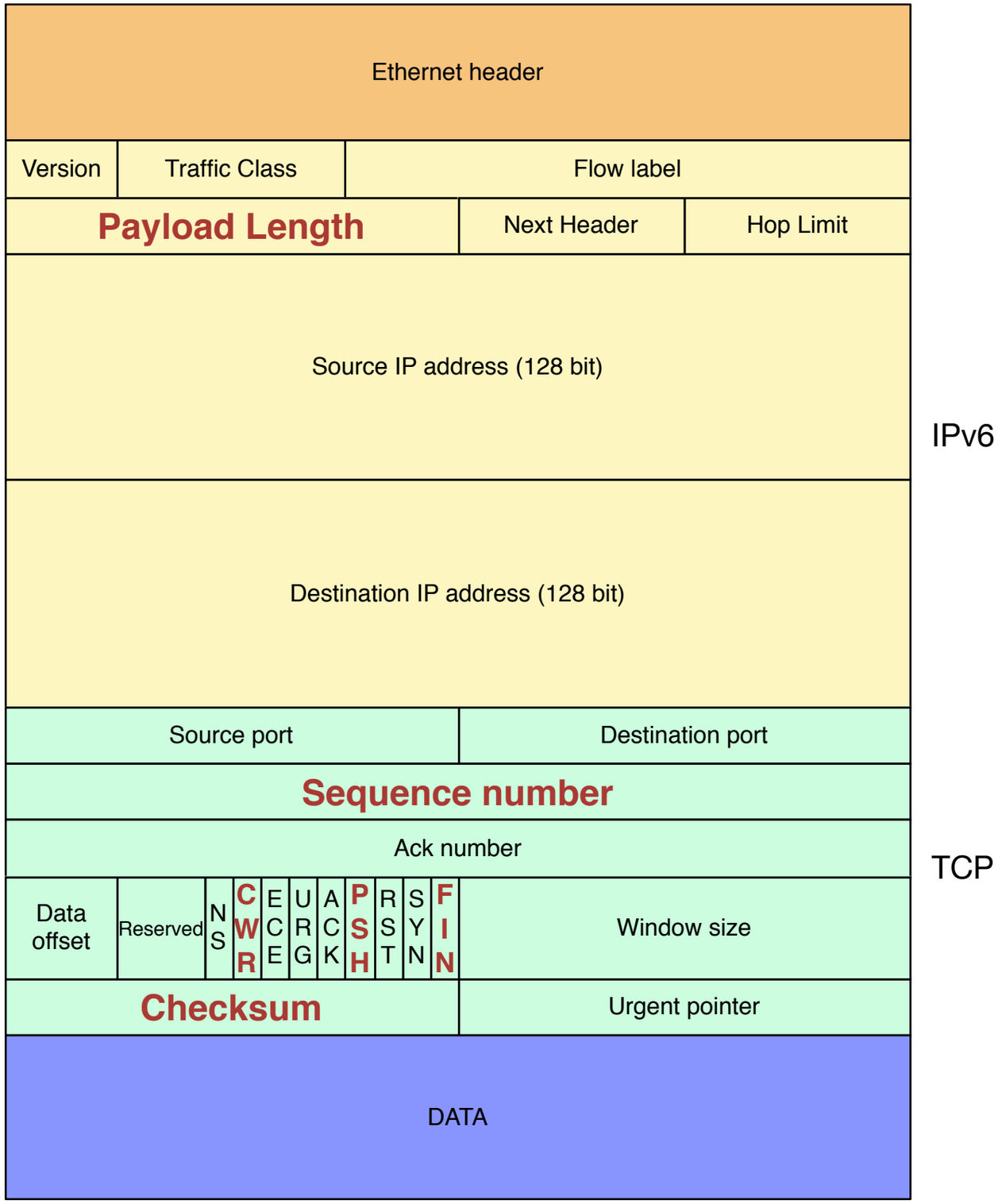
GSO: m_seg()


*m0
hdr_len
mbuf
m_next
m_nextpkt
m_data
m_len
m_flags
m_type
m_pkthdr.len
m_pkthdr.csum_flags
m_pkthdr.csum_data
m_pkthdr.tso_segsz
m_pkthdr.recvif
m_pkthdr.tags
…
m_ext.ext_buf
m_ext.ext_size
m_ext.ext_type
m_ext.ext_free
m_ext.ref_cnt
Ethern Header
IP Header
TCP Header
not used
NULL
54
M_PKTHDR
MT_DATA
64KB
CSUM_GSO
1500
M_EXT
m_seg(struct mbuf *m0, int hdr_len, int mss, …
mss

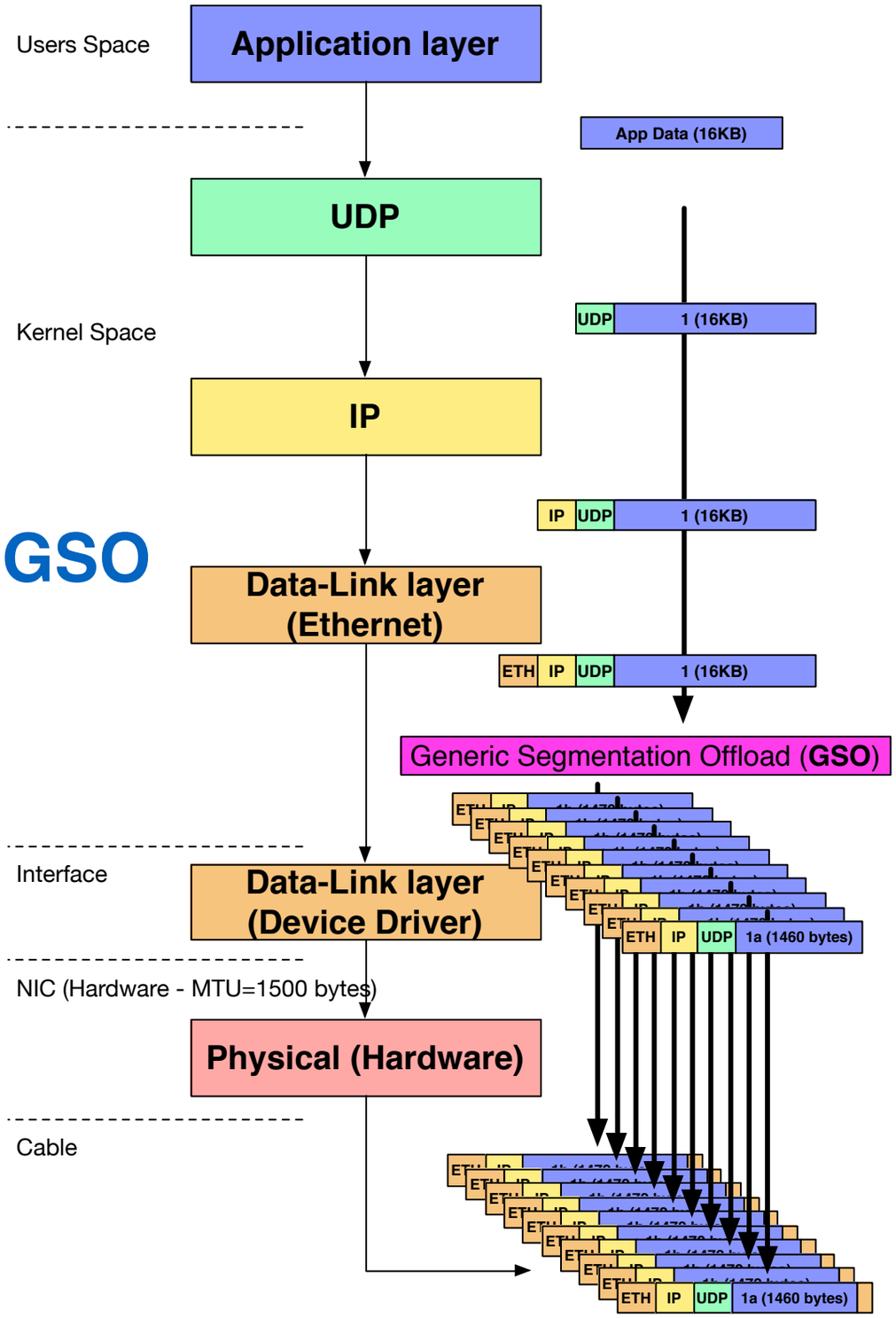
GSO: fix TCP/IPv4 headers



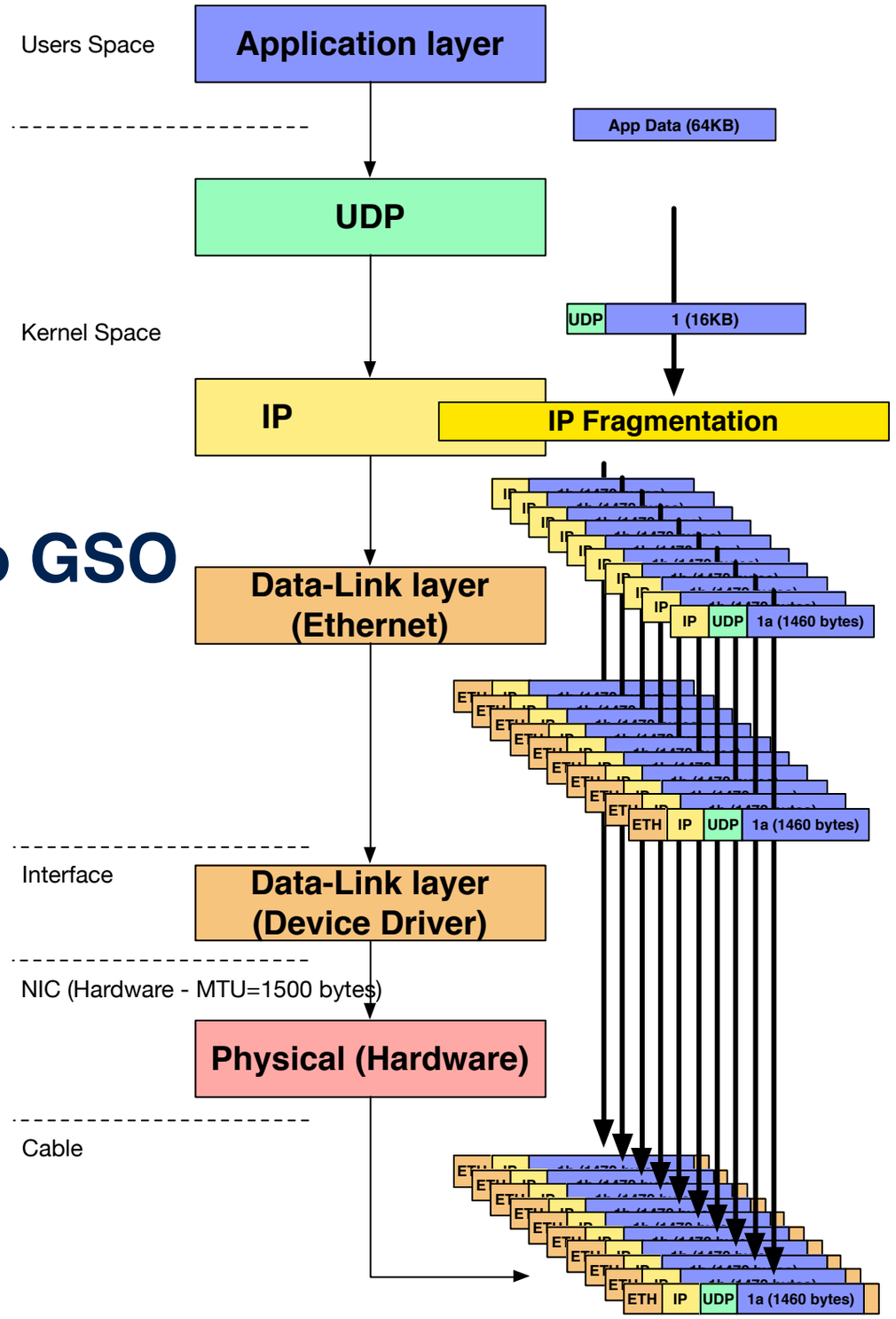
GSO: fix TCP/IPv6 headers



GSO on UDP flow

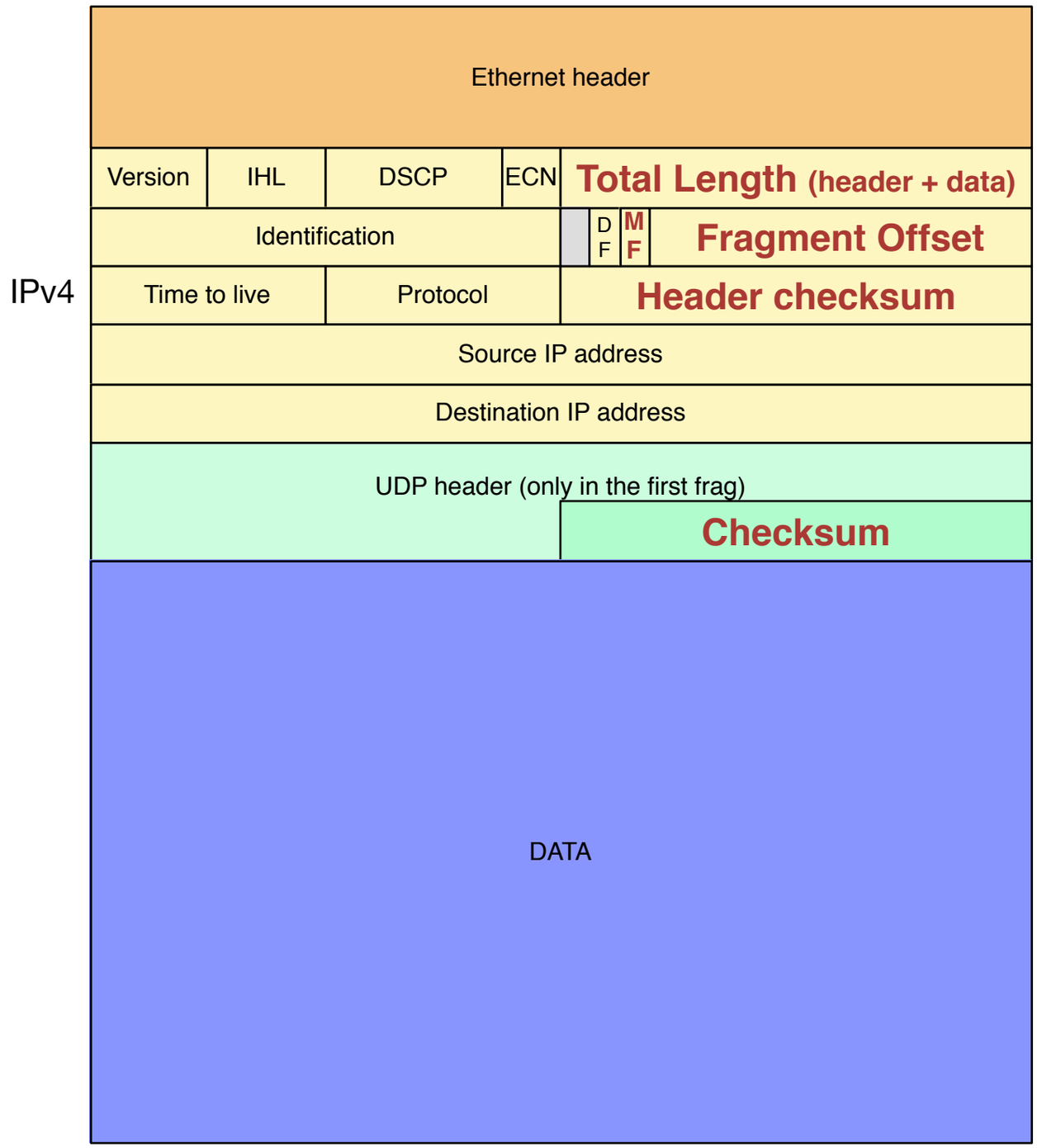


a) GSO

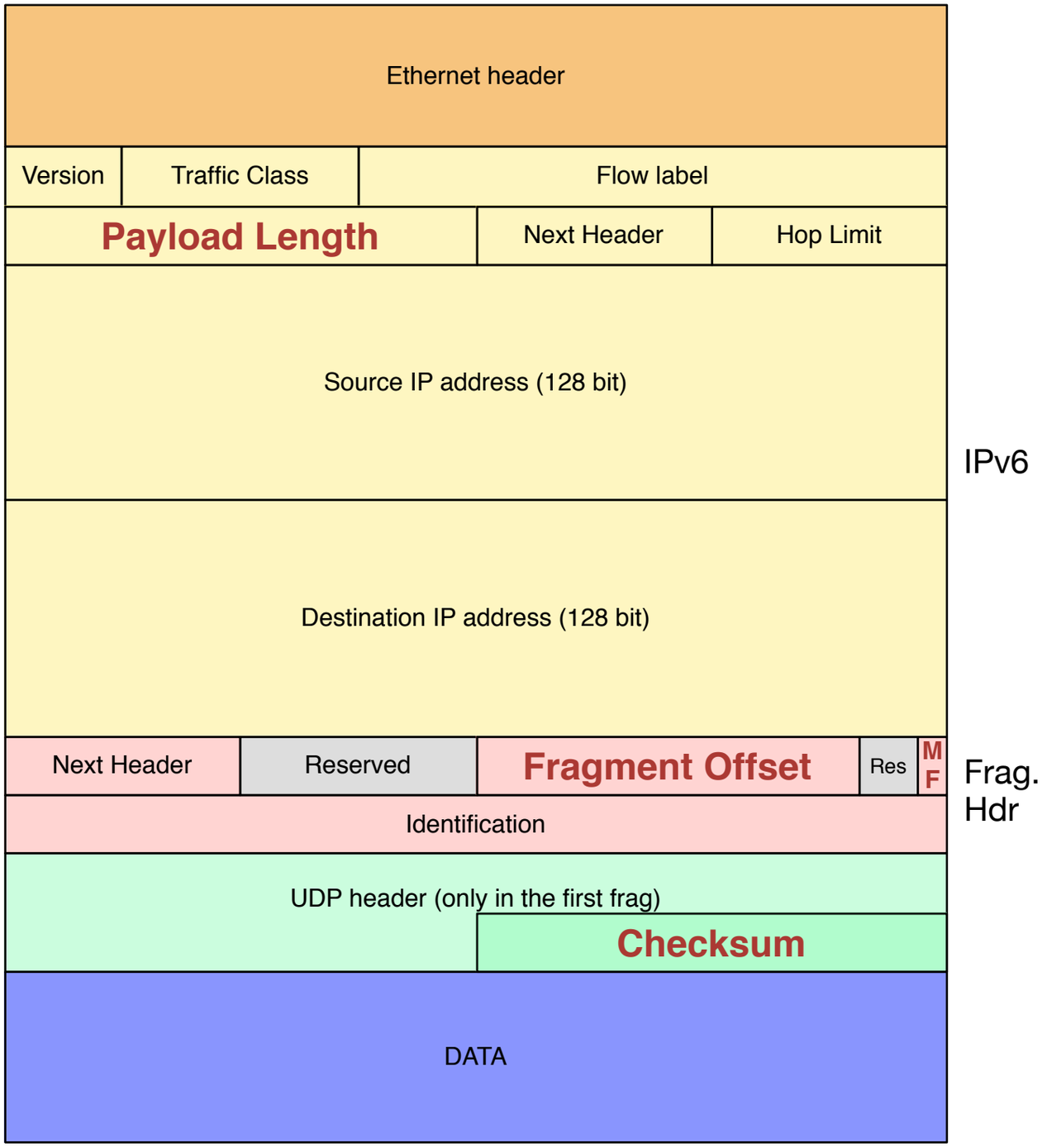


b) no GSO

GSO: fix UDP/IPv4 headers



GSO: fix UDP/IPv6 headers



sysctl

To manage the GSO parameters there are some **sysctl**:

- `net.inet.tcp.gso`
  GSO enable on **TCP** communications (!=0)

- `net.inet.udp.gso`
  GSO enable on **UDP** communications (!=0)

- for each interface:

  - `net.gso.dev."ifname".max_burst`
    GSO burst length limit [default: IP_MAXPACKET=65535]

  - `net.gso.dev."ifname".enable_gso`
    GSO enable on “ifname” interface (!=0)

GSO code

- Kernel patches for FreeBSD-current, FreeBSD 10-stable and FreeBSD 9-stable available at:

  https://github.com/stefano-garzarella/freebsd-gso

- FreeBSD source with GSO available at:

  https://github.com/stefano-garzarella/freebsd-gso-src

- To compile kernel with GSO support:
  - "**options GSO**" in kernel config

GSO patch

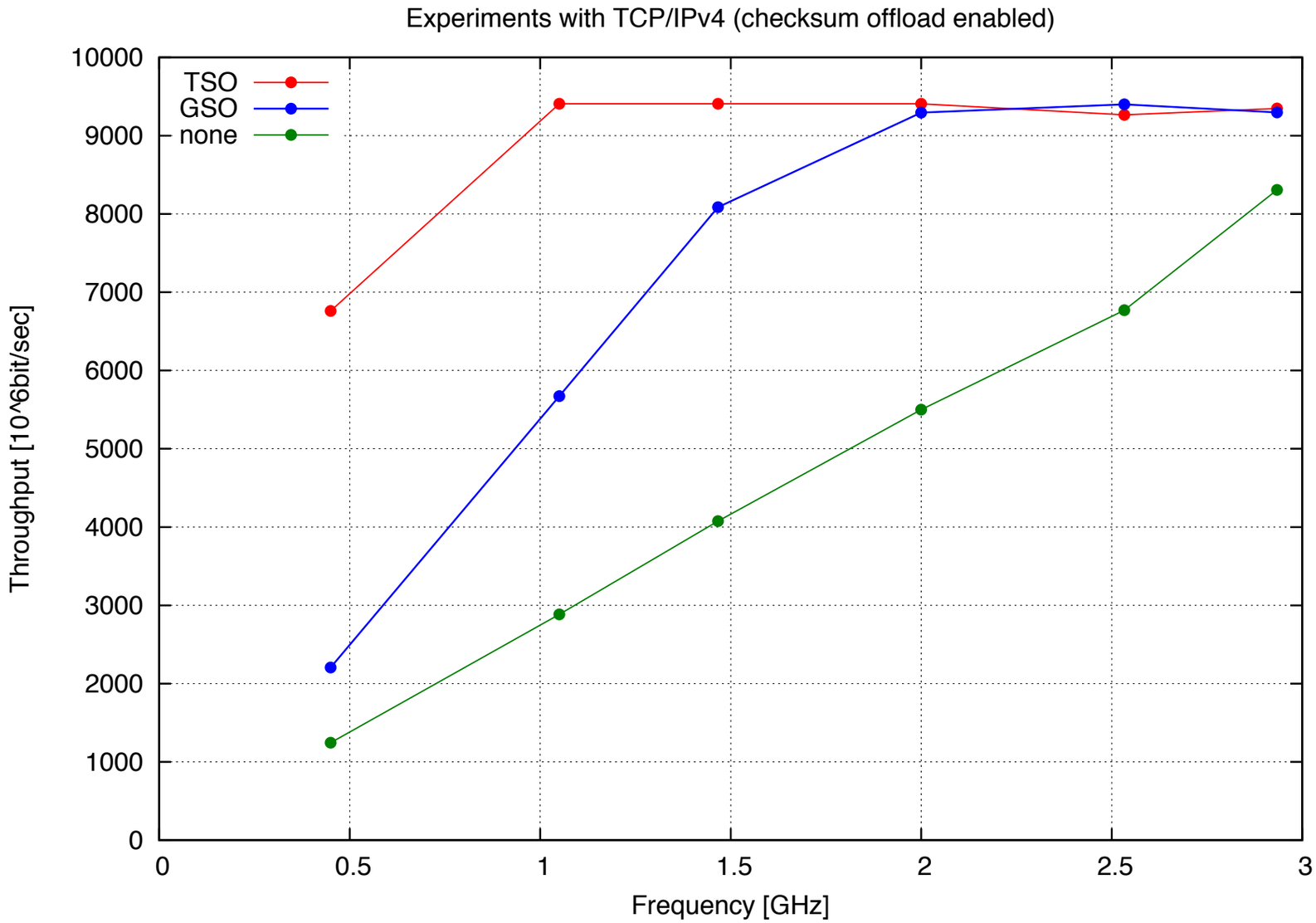
```
diff gso-current

sys/conf/NOTES                |    2 +
sys/conf/files                |    1 +
sys/conf/options              |    1 +
sys/net/gso.c                 | 1020 ++++++++++++++++++++++++++++++++++++++++++++++++++++++
sys/net/gso.h                 |  174 +++++++++
sys/net/if_ethersubr.c        |   31 ++
sys/net/if_var.h              |    1 +
sys/netinet/ip_output.c       |   43 ++-
sys/netinet/tcp_input.c       |   11 +
sys/netinet/tcp_output.c      |   84 ++++-
sys/netinet/tcp_subr.c        |   15 +
sys/netinet/tcp_var.h         |    6 +-
sys/netinet/udp_usrreq.c      |   22 +-
sys/netinet/udp_var.h         |    6 +
sys/netinet6/ip6_output.c     |   51 ++-
sys/netinet6/udp6_usrreq.c    |    8 +
sys/sys/mbuf.h                |    2 +
17 files changed, 1459 insertions(+), 19 deletions(-)
```

Experiments

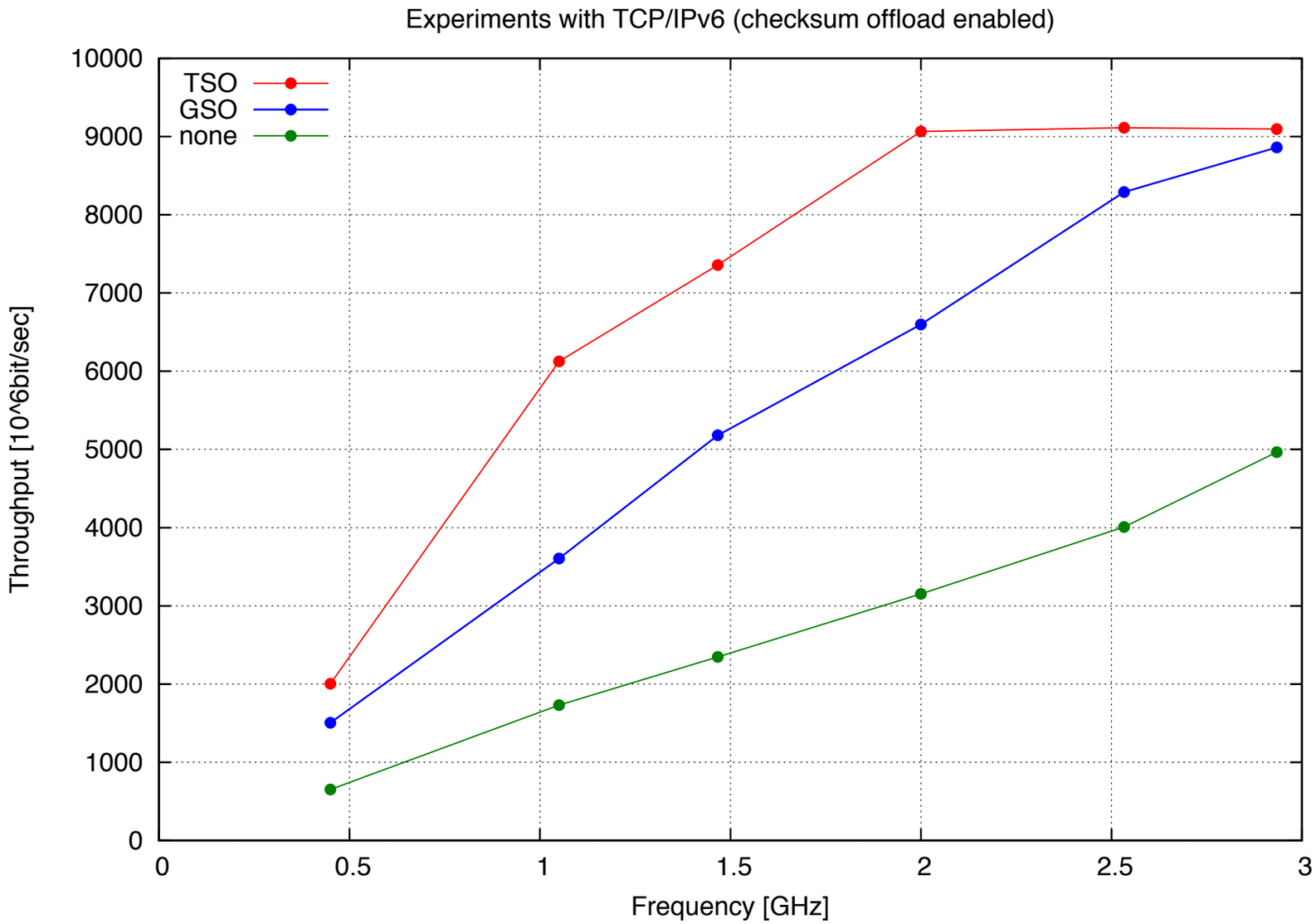
- Sender: CPU i7-870 at 2.93 GHz + Turboboost, Intel 10 Gbit NIC.
- Receiver: CPU i7-3770K at 3.50GHz + Turboboost, Intel 10 Gbit NIC.
    - (RSC/LRO)-enabled (otherwise TSO/GSO are ineffective)
- Benchmark tool: netperf 2.6.0

Results TCP/IPv4



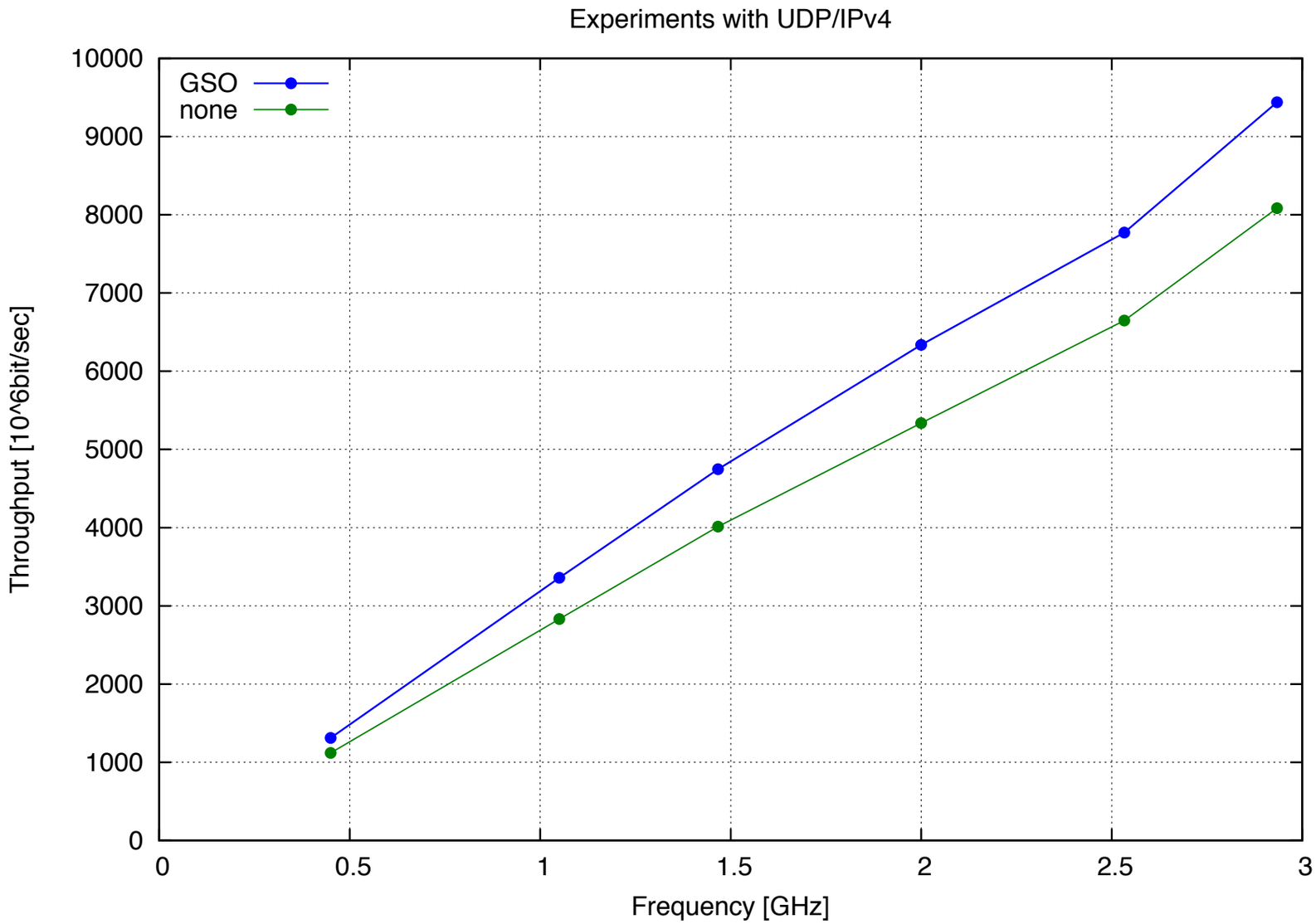
Freq. [GHz]	TSO	GSO	none	Speedup GSO - none
2.93	9347	9298	8308	**12%**
2.53	9266	9401	6771	**39%**
2.00	9408	9294	5499	**69%**
1.46	9408	8087	4075	**98%**
1.05	9408	5673	2884	**97%**
0.45	6760	2206	1244	**77%**

Results TCP/IPv6



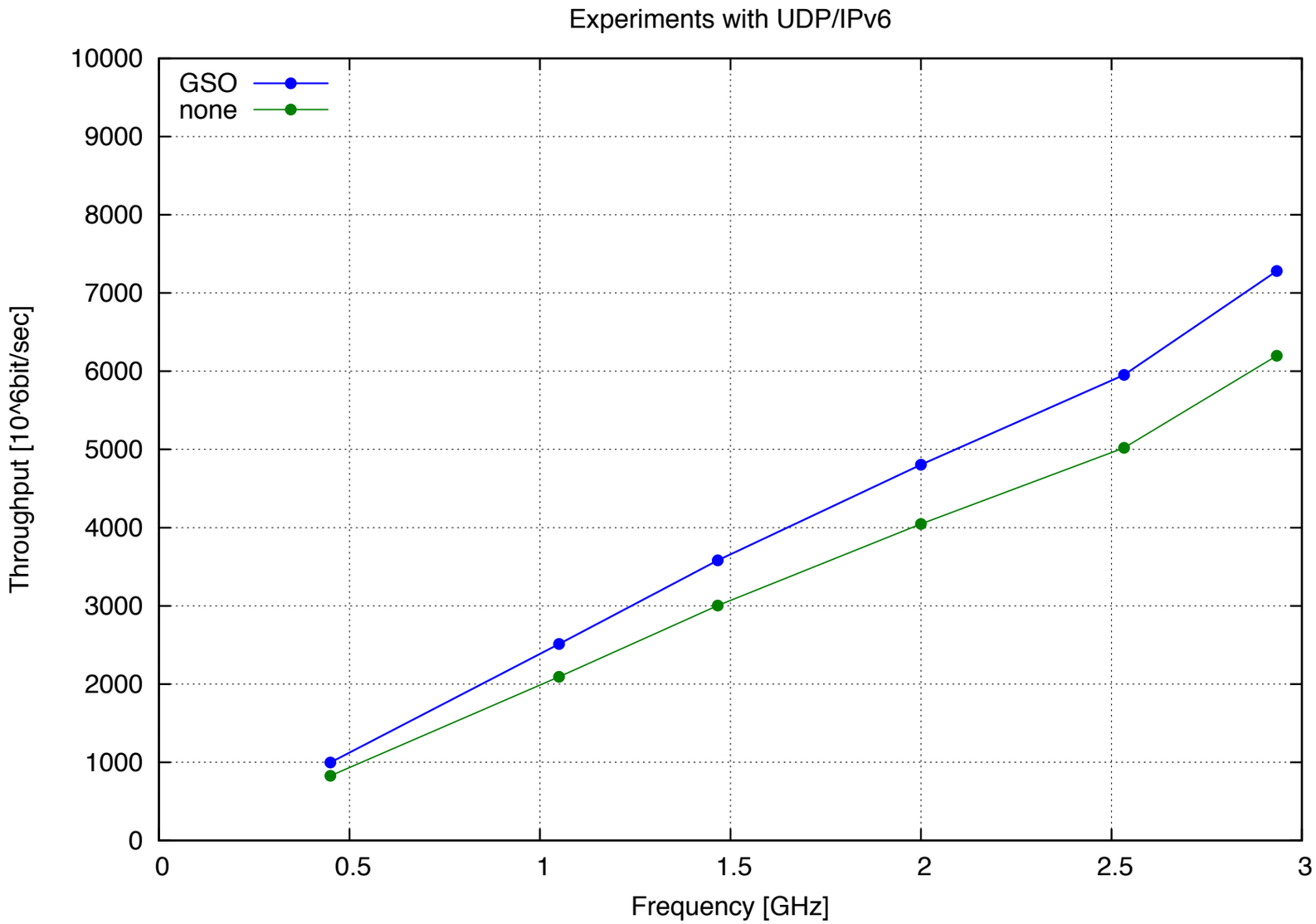
Freq. [GHz]	TSO	GSO	none	Speedup GSO - none
2.93	9097	8861	4966	**78%**
2.53	9113	8290	4008	**107%**
2.00	9066	6599	3152	**109%**
1.46	7357	5180	2348	**121%**
1.05	6125	3607	1732	**108%**
0.45	2005	1505	651	**131%**

Results UDP/IPv4



Freq. [GHz]	GSO	none	Speedup GSO - none
2.93	9440	8084	17%
2.53	7772	6649	17%
2.00	6336	5338	19%
1.46	4748	4014	18%
1.05	3359	2831	19%
0.45	1312	1120	17%

Results UDP/IPv6



Freq. [GHz]	GSO	none	Speedup GSO - none
2.93	7281	6197	**17%**
2.53	5953	5020	**19%**
2.00	4804	4048	**19%**
1.46	3582	3004	**19%**
1.05	2512	2092	**20%**
0.45	998	826	**21%**

Future works

- More performance measurements
- Optimize code paths
- Add support to new protocols (SCTP, …)

Thank you!