Optimizing UDP for content delivery: GSO, pacing and zerocopy

Willem de Bruijn
willemb@google.com

Workload: QUIC

>35 % of Google egress

- stream multiplexing, low latency connection establishment, …
- 2x higher cycle/Byte than TCP
- serving ~10K concurrent 1 MBps flows per server

"The QUIC Transport Protocol", SIGCOMM 2017
"QUIC - Developing and Deploying a TCP Replacement for the Web", netdevconf 0x12
"Live encoder settings, bitrates, and resolutions", support.google.com/youtube/answer/2853702

UDP

unreliable datagrams.. but also:

rapid experimentation & deployment

- widely available
- no superuser privileges
- middlebox support
- thin service, so highly extensible

UDP cycle efficiency

	calls/s	Mcycles/s	Speed-up (%)
TCP	19040	618	**487**
UDP	812000	2801	100

`tools/testing/selftests/net/udpgso_bench_tx`

UDP cycle efficiency

	calls/s	Mcycles/s	Speed-up (%)
TCP no-segs	19040	2800	100
TCP gso	19040	1856	162
TCP tso	19040	618	487
UDP	812000	2801	100

```
ethtool -k $DEV tso off gso on
```

Optimizing the UDP serving stack

- UDP_SEGMENT
- GSO_PARTIAL
- MSG_ZEROCOPY
- SO_TXTIME
- UDP_GRO

Work of many others

- Alexander Duyk
- Boris Pismenny
- Edward Cree
- Eric Dumazet
- Jesus Sanchez-Palencia
- Paolo Abeni
- Steffen Klassert
- ...

GSO:

fewer, larger packets

UDP GSO: segmentation offload

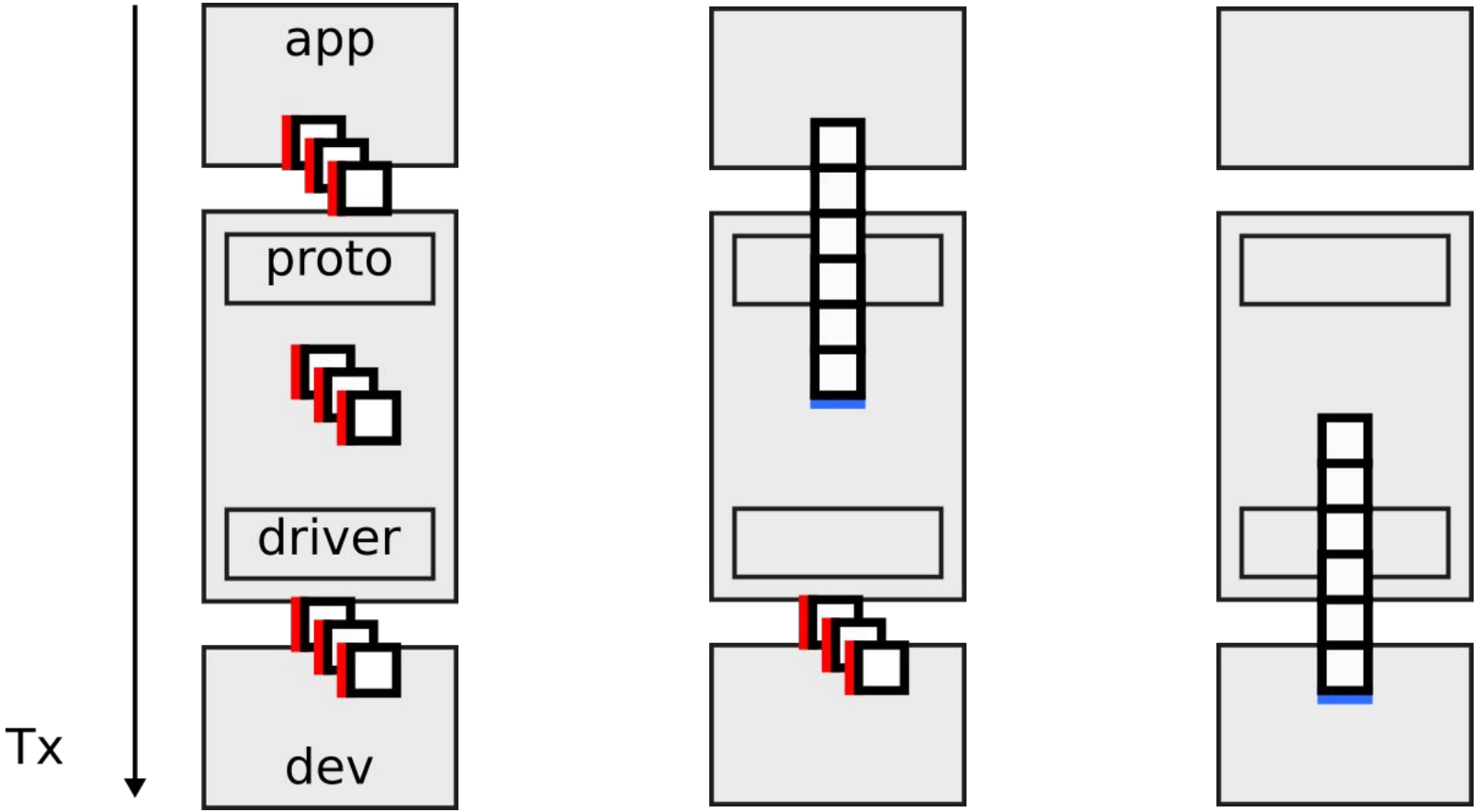
virtual high MTU link

1500 - 40 - 8 == 1452 B payload per send

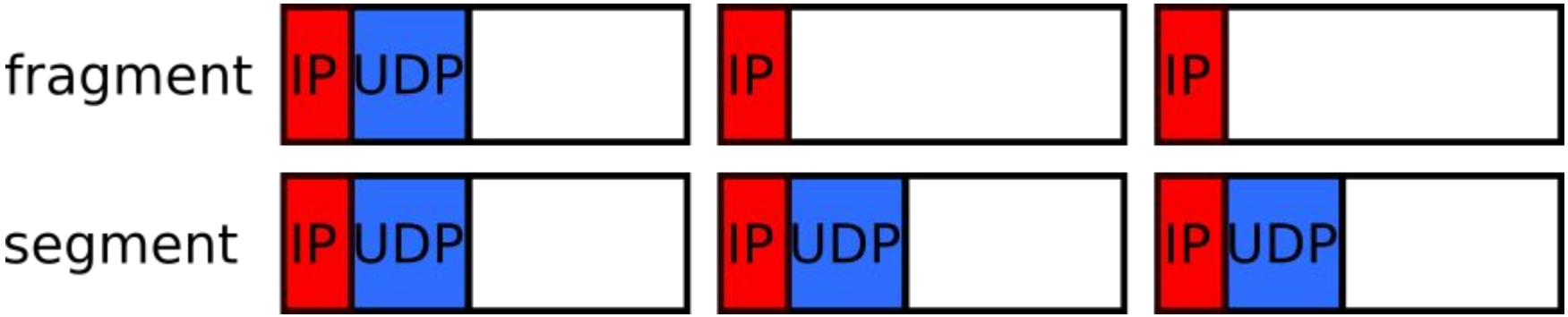
65535 - 8 == 65527

~45x reduction in stack traversals

UDP GSO: stack traversal



UDP GSO != UFO



UDP GSO: interface


```
int gso_size = ETH_DATA_LEN - sizeof(struct ipv6hdr) - sizeof(struct udphdr);
if (setsockopt(fd, SOL_UDP, UDP_SEGMENT, &gso_size, sizeof(gso_size)))
error(1, errno, "setsockopt udp segment");

cm = CMSG_FIRSTHDR(&msg);
cm->cmsg_level = SOL_UDP;
cm->cmsg_type = UDP_SEGMENT;
cm->cmsg_len = CMSG_LEN(sizeof(uint16_t));
*((uint16_t *) CMSG_DATA(cm)) = gso_size;
ret = sendmsg(fd, &msg, 0);
```

UDP GSO: evaluation

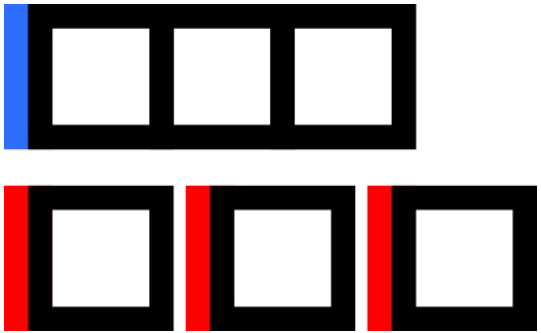
	calls/s	Mcycles/s	Speed-up (%)
TCP no-segs	19040	2800	100
TCP gso	19040	1856	162
TCP tso	19040	618	487
UDP	812000	2801	100
UDP gso	18248	1726	174

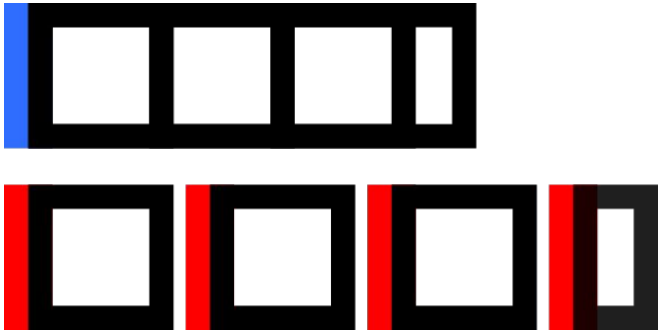
UDP GSO: evaluation

	calls/s	Mcycles/s	Speed-up (%)
TCP no-segs	19040	2800	100
TCP gso	19040	1856	162
TCP tso	19040	618	487
UDP	812000	2801	100
UDP gso	18248	1726	174
UDP lso	..	..	..

"Udp segmentation offload", netdevconf 0x12

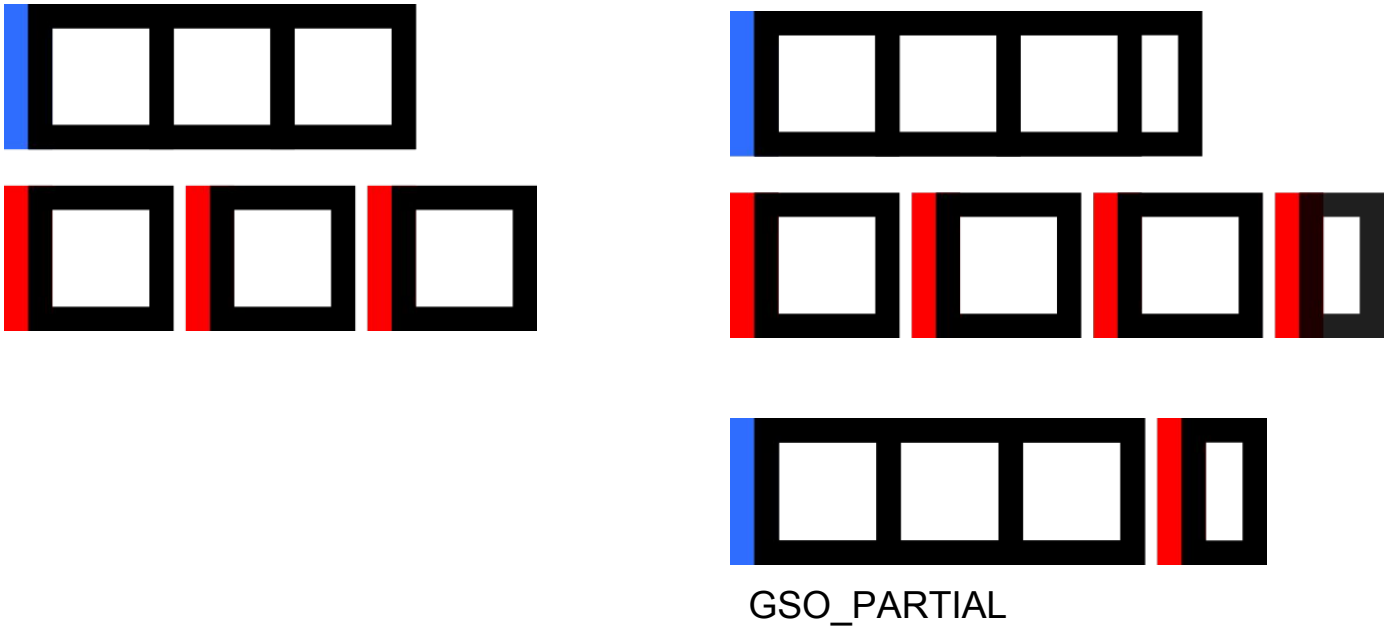
UDP GSO: hardware





"Encapsulation Offloads: LCO, GSO_PARTIAL, [..]", netdevconf 1.2

UDP GSO: hybrid



"Encapsulation Offloads: LCO, GSO_PARTIAL, [..]", netdevconf 1.2

UDP GSO: implementation details

- choosing `gso_size`
  - ETH_DATA_LEN
  - path MTU (IP_MTU_DISCOVER)

- choosing number of segments
  - fit in network layer
  - <= UDP_MAX_SEGMENTS
  - > gso_size

- checksum offload
  - csum_and_copy_from_user

`tools/testing/selftests/net/udpgso`

MSG_ZEROCOPY:

tx copy avoidance

MSG_ZEROCOPY

```
perf record netperf -t TCP_STREAM -H $host

Samples: 42K of event 'cycles', Event count (approx.): 21258597313
 79.41%         33884  netperf  [kernel.kallsyms]  [k] copy_user_generic_string
  3.27%          1396  netperf  [kernel.kallsyms]  [k] tcp_sendmsg
  1.66%           694  netperf  [kernel.kallsyms]  [k] get_page_from_freelist
  0.79%           325  netperf  [kernel.kallsyms]  [k] tcp_ack
  0.43%           188  netperf  [kernel.kallsyms]  [k] __alloc_skb
```

"sendmsg copy avoidance with MSG_ZEROCOPY", netdevconf 2.1

MSG_ZEROCOPY: evaluation

		Copy
	copy %	Mcyc/s
TCP	26.7	618
UDP	**3.11**	2800

MSG_ZEROCOPY: evaluation

		Copy
	copy %	Mcyc/s
TCP	4.35	2800
TCP gso	10.3	1856
TCP tso	26.7	618
UDP	3.11	2800
UDP gso	13.4	1727
UDP gso (CT)	21.2	1916

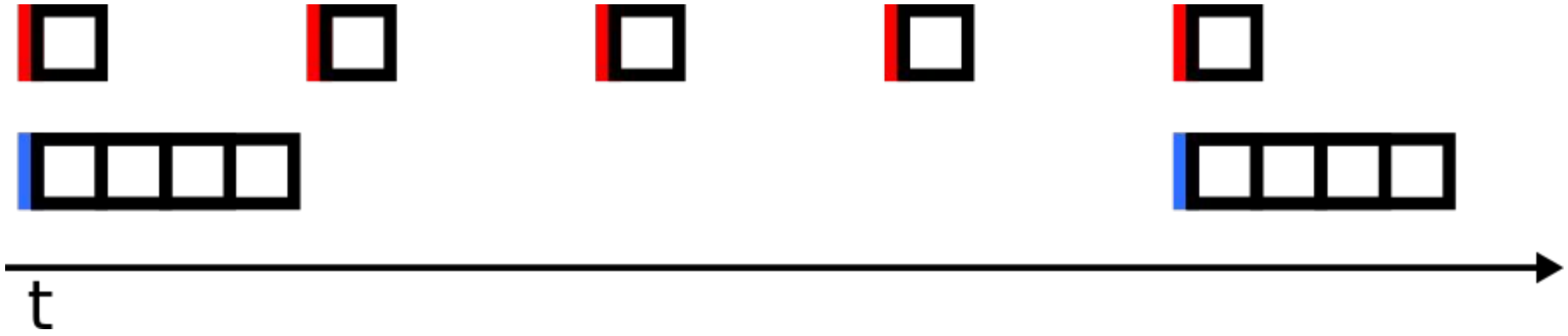
MSG_ZEROCOPY: evaluation

		Copy	Zerocopy	Speed-up
	copy %	Mcyc/s	Mcyc/s	%
TCP	4.35	2800	2800	100
TCP gso	10.3	1856	1704	109
TCP tso	26.7	618	425	145
UDP	3.11	2800	2800	100
UDP gso	13.4	1727	1690	102
UDP gso (CT)	21.2	1916	1694	113

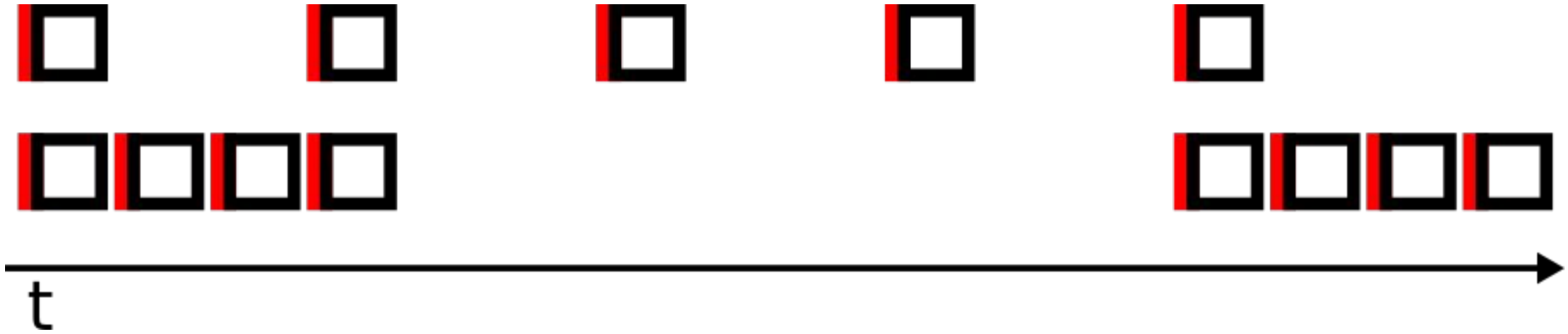
Pacing offload:

avoid retransmits

Pacing: larger bursts with GSO



Pacing: larger bursts with GSO



Pacing

- Bursts
  - -> higher drops
  - -> higher retransmit
  - -> higher cyc/B
- 10k clients at 1MBps
  - RR? 1MB in 100 usec
- Pace: send at 1 msec interval
- Pacing offload: reduce jitter, reduce cycle/B
    - SO_MAX_PACING_RATE
    - SCH_FQ

Pacing: SO_TXTIME interface

```
const int flags = 0;

setsockopt(fd, SOL_SOCKET, SO_TXTIME, &flags, sizeof(flags))

clock_gettime(CLOCK_TAI, &ts);
uint64_t txtime = ts.tv_sec * 1000000000ULL + ts.tv_nsec + txdelay_ns;

cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_TXTIME;
cmsg->cmsg_len = CMSG_LEN(sizeof(__u64));
*((__u64 *) CMSG_DATA(cmsg)) = txtime;
```

"Scheduled packet transmission: Etf", https://lwn.net/Articles/758592/

Pacing & GSO

- Conflicting goals
  - maximize batching
  - send at msec interval

- Pace at millisecond granularity
  - 1 MBps*
  - 1KB per msec
  - < 1 MSS

Pacing & GSO: evaluation

Pacing interval (msec)	CPU time %	Loss %
1	100	100
2	92	103
4	88	110
8	84	117

UDP_GRO:

batch receive

UDP GRO

- Inverse operation
  - larger, fewer packets
  - forwarding to GSO
  - local delivery
    - large packets
    - transparent
      - segment
      - frag list
      - netfilter redirect
  - Listification

"udp: implement gro support", https://lwn.net/Articles/768995/
"Handle multiple received packets at each stage", http://patchwork.ozlabs.org/project/netdev/list/?series=53249&state=*

UDP GRO: interface

```
setsockopt(fd, IPPROTO_UDP, UDP_GRO, &enable, sizeof(enable));

recvmsg(fd, &msg, 0);

for (cm = CMSG_FIRSTHDR(&msg); cm; cm = CMSG_NXTHDR(&msg, cmsg))
        if (cm->cmsg_level == SOL_UDP && cm->cmsg_type == UDP_GRO)
                gso_size = *(uint16_t *) CMSG_DATA(cmsg);
```

UDP GRO: evaluation

	Gbps	calls/s	Mcycles/c	Speed-up (%)
UDP	798	568000	3564	100
UDP GRO	1022	40250	2498	182

Caveat: no sufficient packet trains across WAN in practice?

Summary

UDP_SEGMENT
GSO_PARTIAL
MSG_ZEROCOPY
SO_TXTIME
UDP_GRO

Questions?