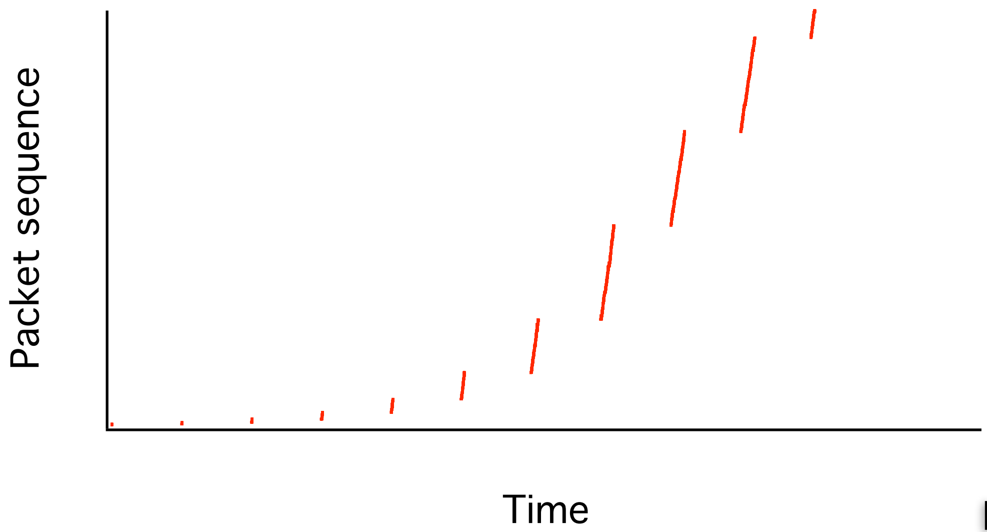
Transmission Timer Approach for Rate Based Pacing TCP with Hardware Support

Katsushi Kobayashi
ikob@koganei.wide.ad.jp
NICT, JAPAN

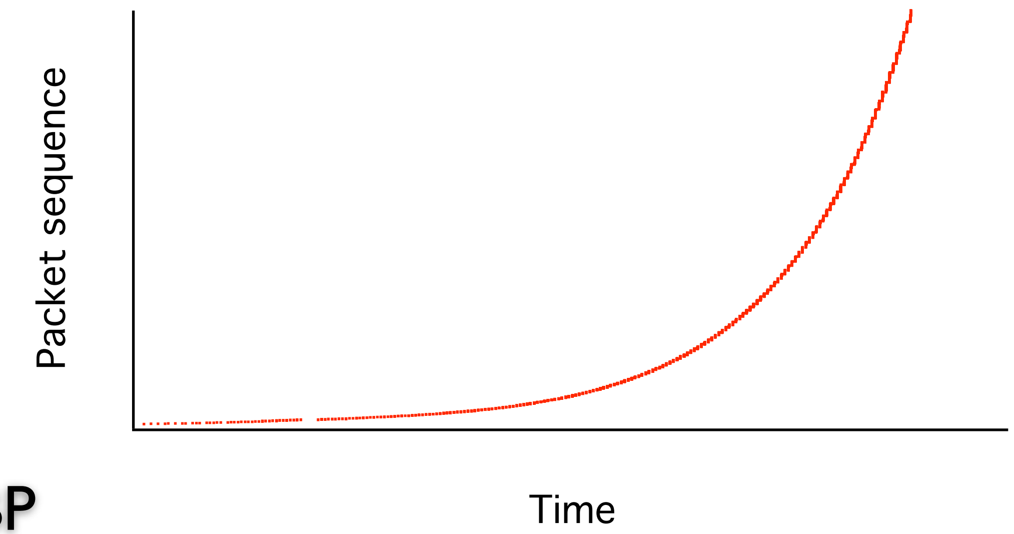
NiCT

TCP slow-start burstiness and RBP

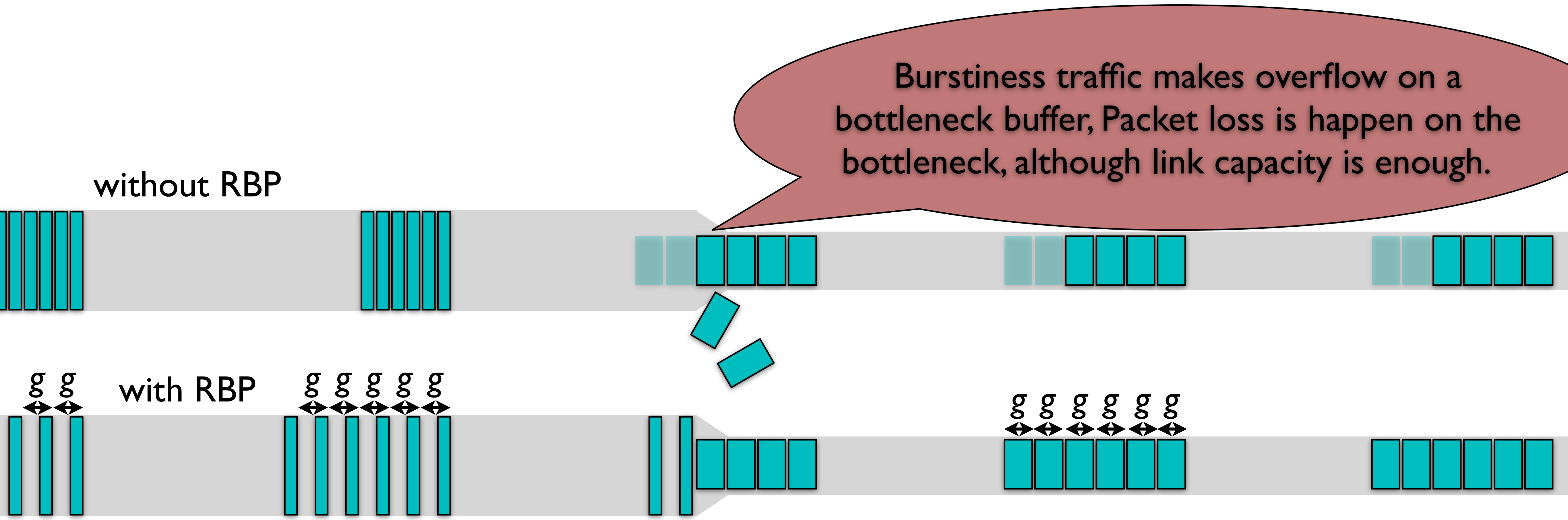


Interframe gap (*g*):

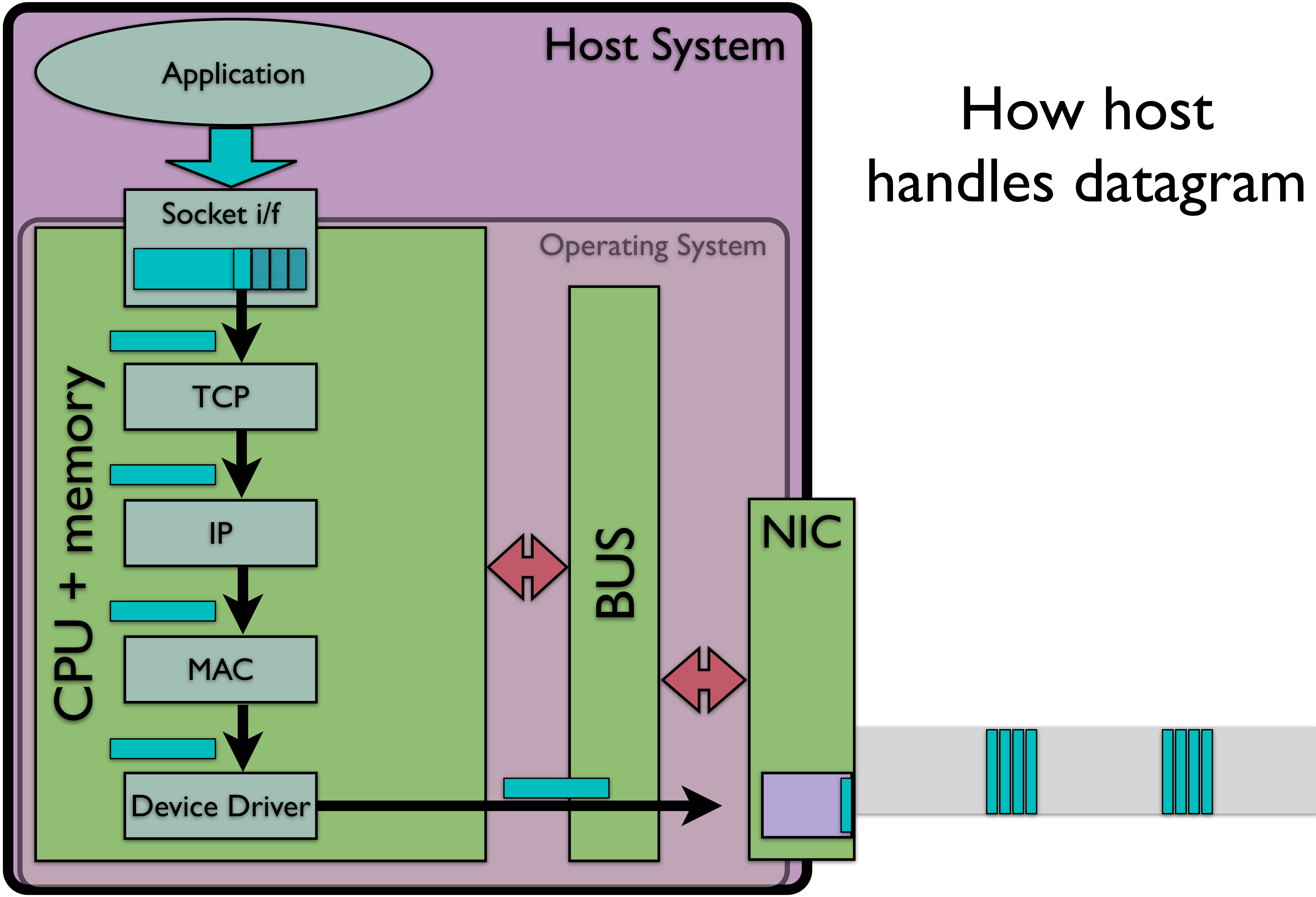
g = RTT x MSS /cwnd



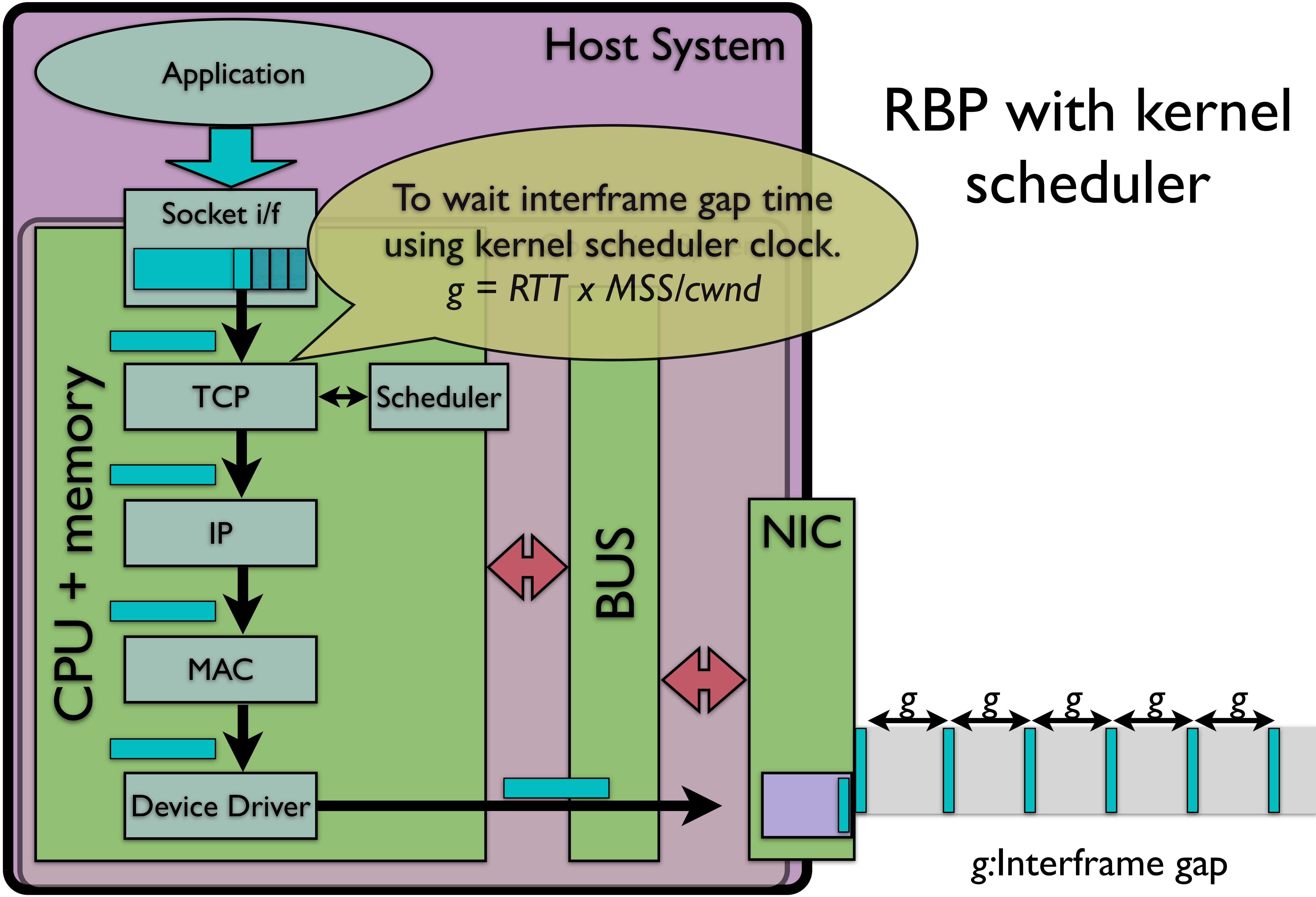
mitigate burstiness with RBP



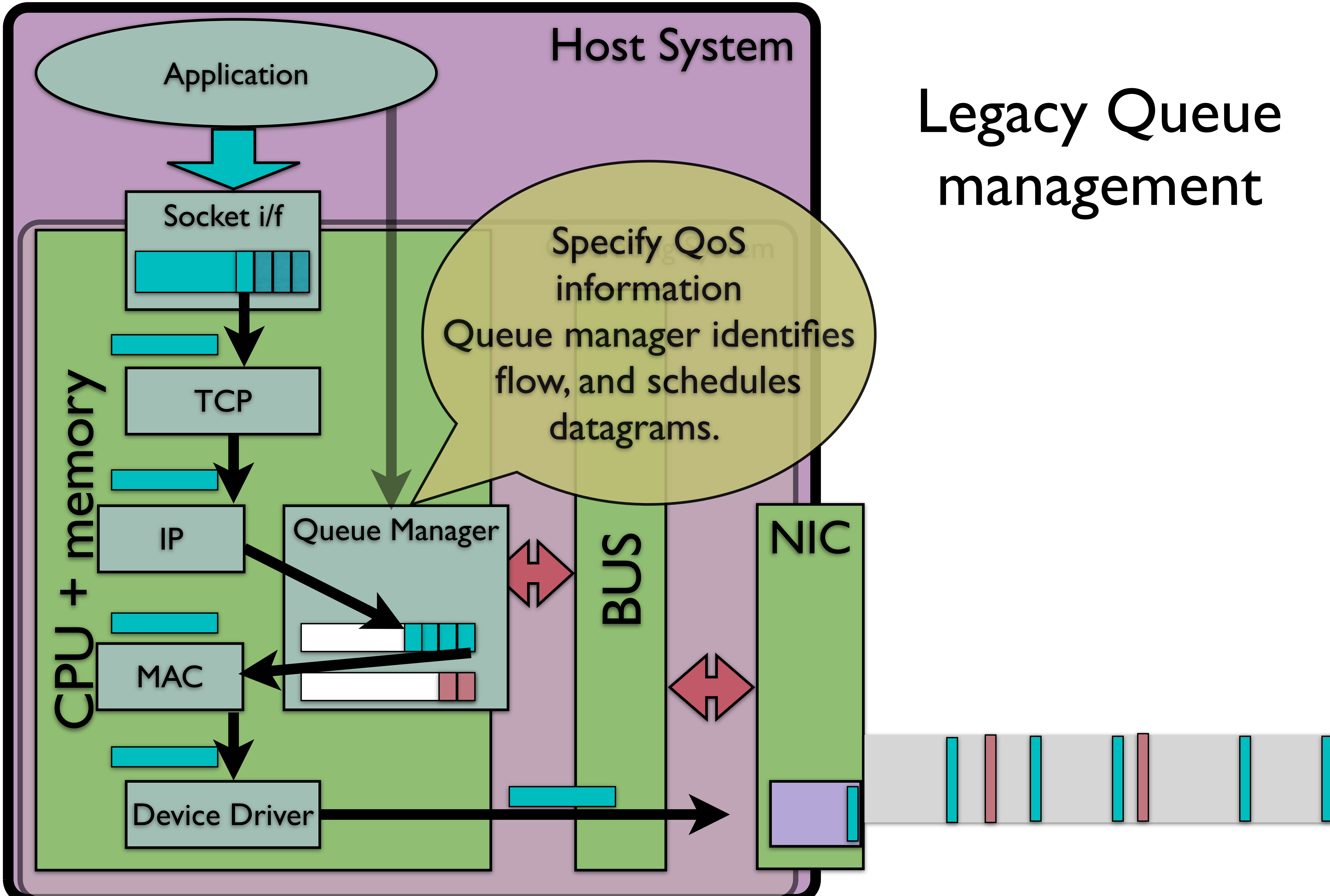
How host handles datagram



RBP with kernel scheduler



Legacy Queue management



RBP TCP with OS scheduler

- Interframe gap@higher bandwidth communication
  << OS scheduler tick granularity as 1 - 10 m sec.

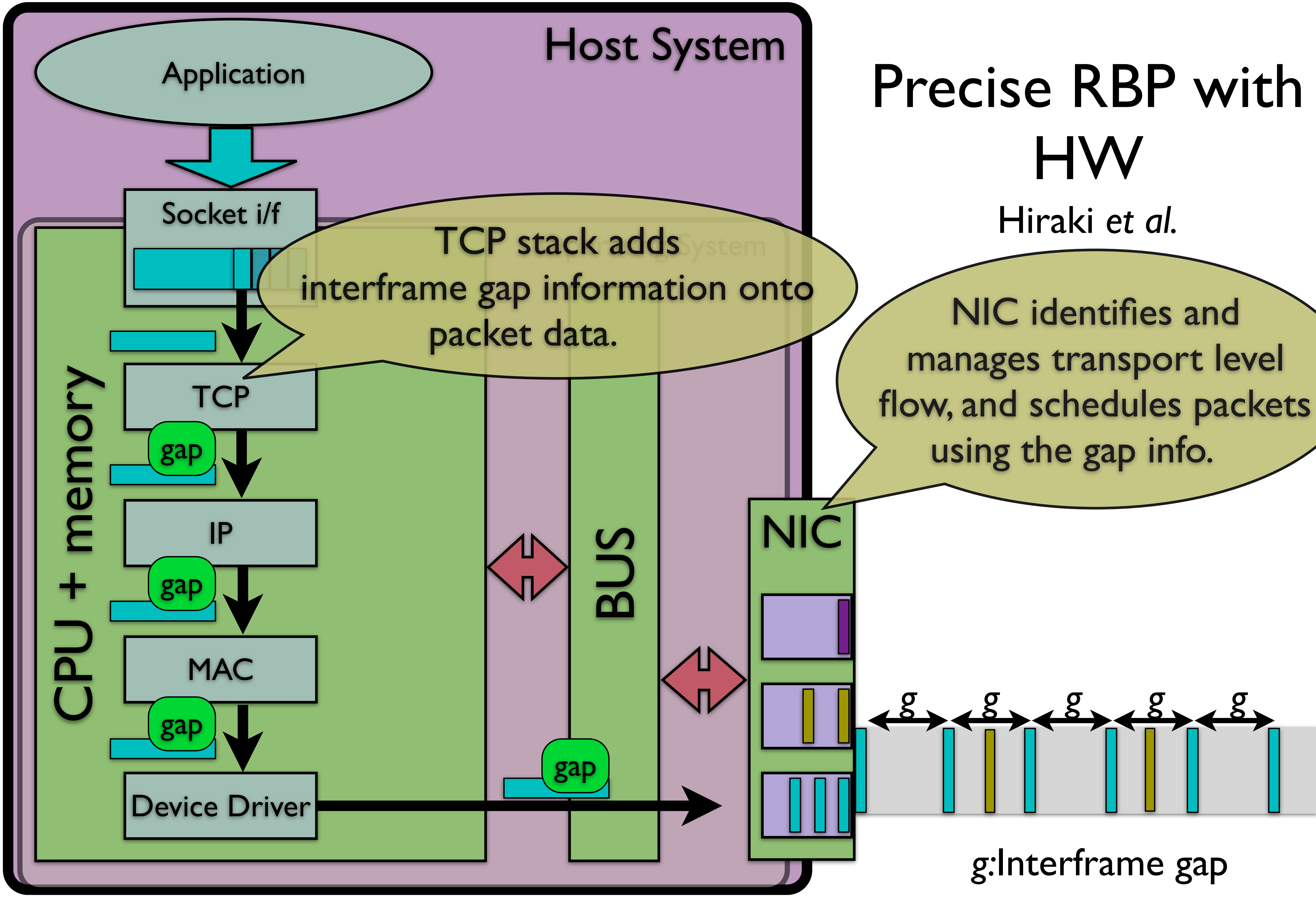
- Precise rate based pacing mechanism not to rely on kernel scheduler is required.

Interframe gap@1500Bytes packet size

Bandwidth	Interframe gap
100Kbps	120 msec.
1Mbps	12 msec.
10Mbps	1.2 msec.
100Mbps	120 u sec.
1Gbps	12 u sec.
10Gbps	1.2 u sec.

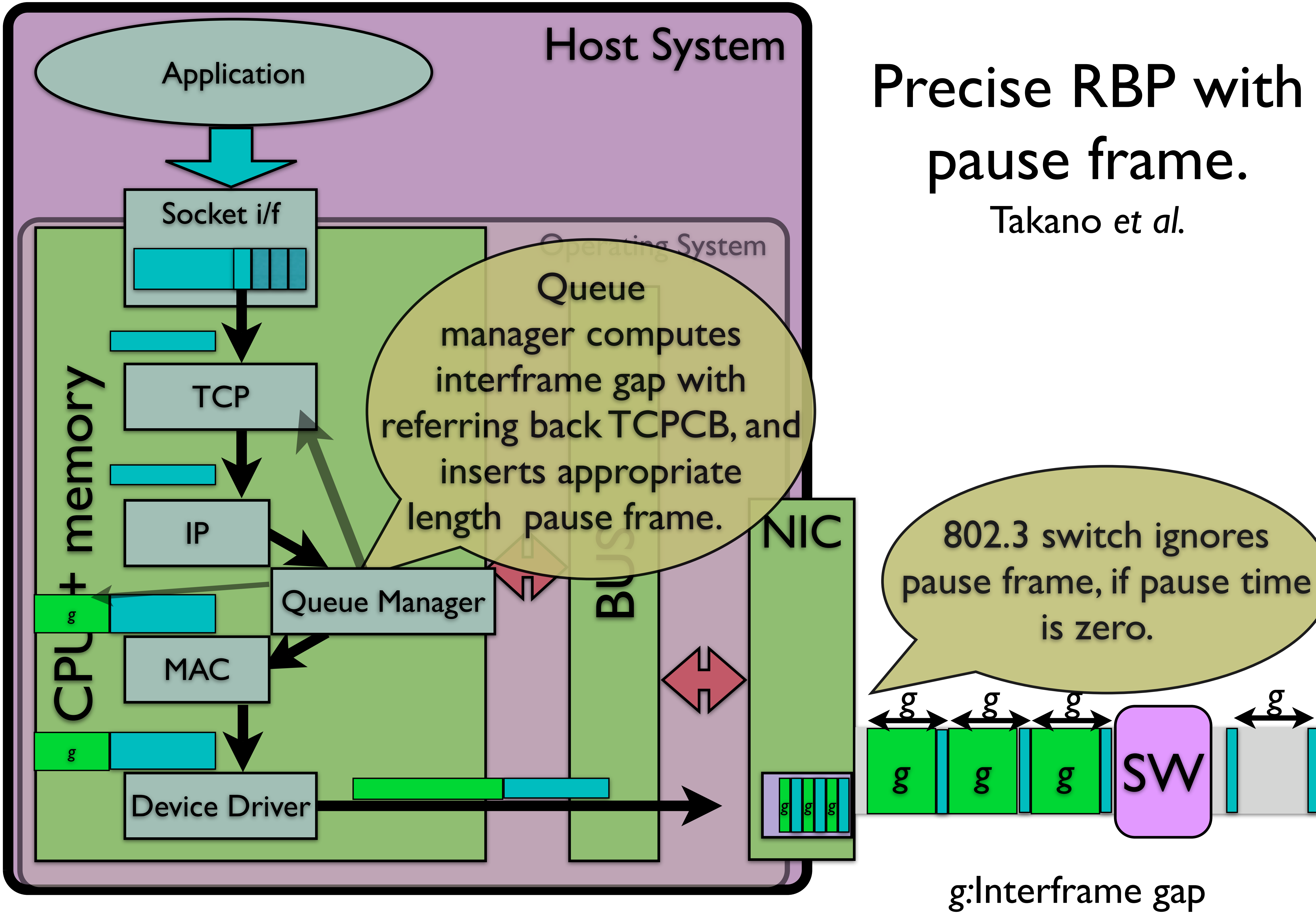
Precise RBP with HW

Hiraki *et al.*



Precise RBP with pause frame.

Takano *et al.*

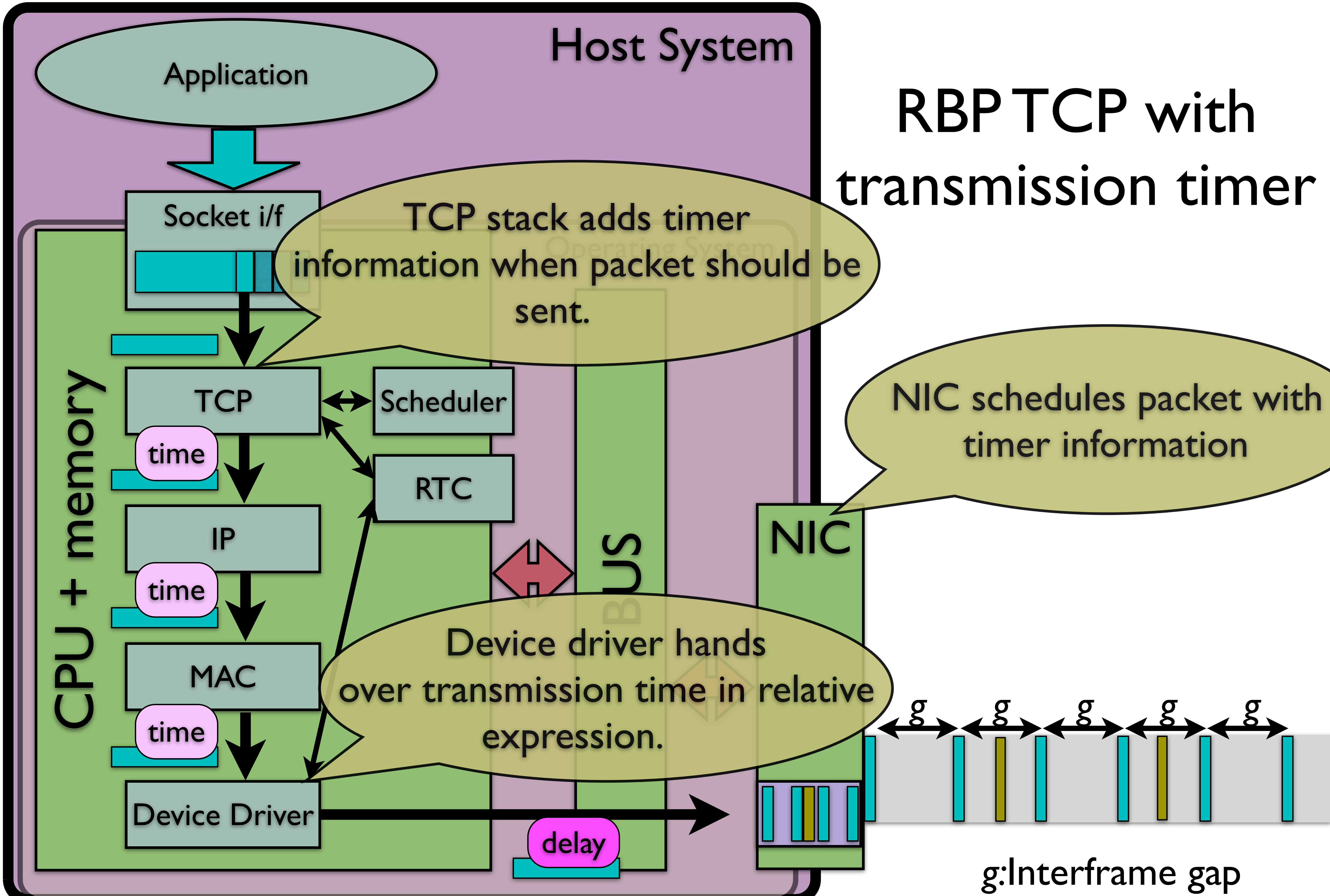


Requirement for precise packet pacing

- Precise packet pacing more than OS scheduler granularity
- Follow frequent interframe gap (rate) changing
- Re-order packet at sender host even on the same connection stream.
- Free from upper layer

- Not depend specific transport protocol.

RBP TCP with transmission timer



NIC implementation

- Intel IXP2400 NP (Radisys ENP2611) act as PCI GbE NIC
  - 3 output queues: transmission timer, high priority, bast effort
  - timer value (delay) specified in DMA descriptor by device driver
- up to 100 milli second delay using 2 level scheduler
  - 4,096 ring bin slots, 27 micro sec./slot
  - NP scheduler, 26.7 nano sec. granularity

RBP TCP implementation

- FreeBSD 4.10
- Add transmission time attribute to mbuf
- Add transmission time field of last segment to tcpcb
- Precise RTT measurement for computing interframe_gap :

  interframe_gap = RTT x MSS/cwnd;
- Estimate tranmission_time for each segment:

  tranmission_time

  = interframe_gap + transmission_time of previous segment;
- Decide whether send it, or postpone next events:

```
if (transmission_time > current_time() + scheduler_tick)
    tsleep(); sleep until next scheduler interrupt      ← RBP with Software
else{
    update mbuf with transmission_time;                 ← RBP with HW
    ip_output();
}
```

NICT

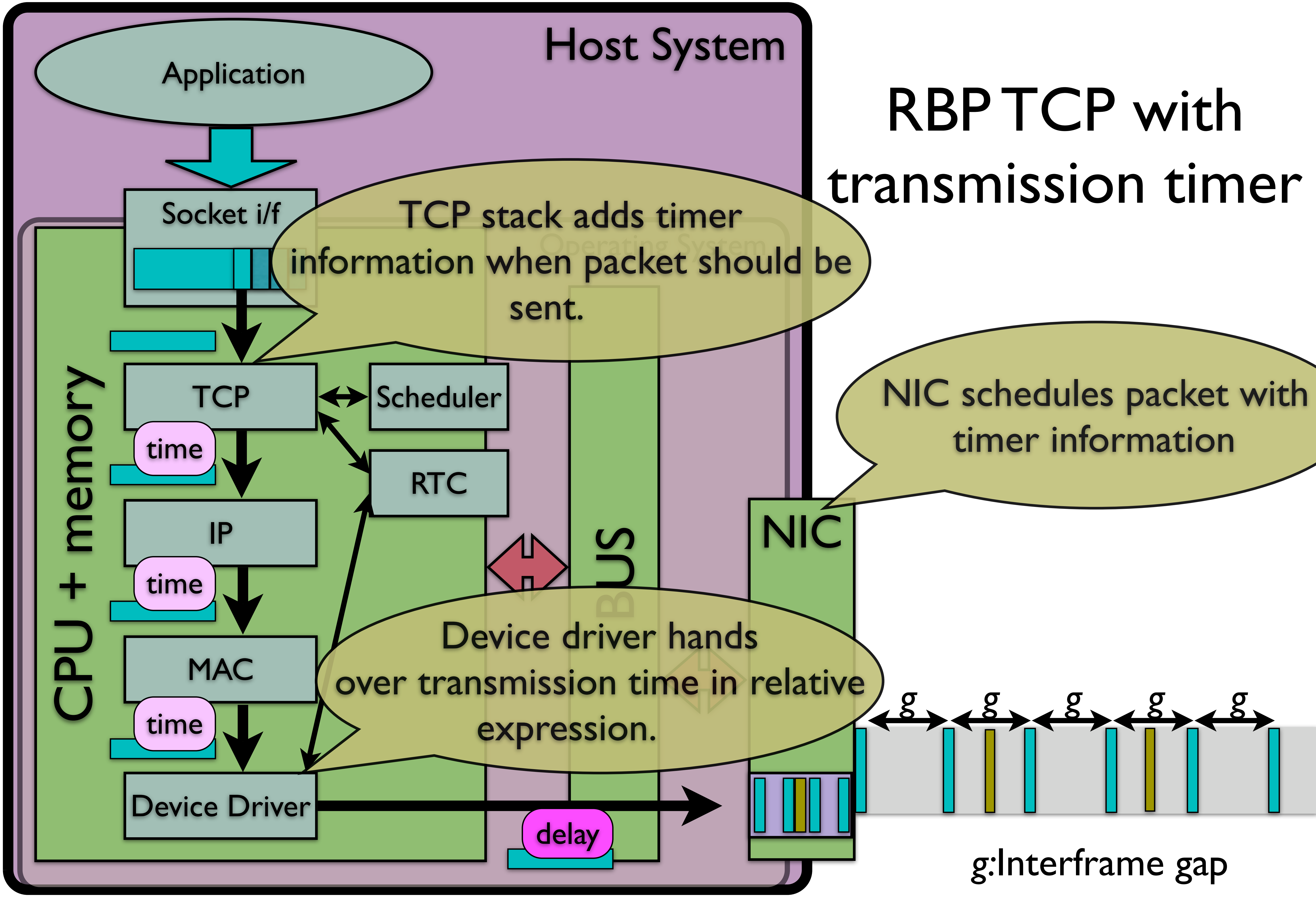
RBP TCP implementation (cont'd)

- Mainly modify tcp_output(), not to modify IP layer.
- Less than 1,000 line hacking in TCP
  - more than 30,000 is TCP code in FreeBSD

- FreeBSD TCP stack does not record the buffer pointer last transmitted.
  - Buffer traverse process is required at every packet.
  - Suppress TCP throughput on large *cwnd*.
  - borrowed from NetBSD 2.0 code.

- Too many referring RTC
  - Suppress TCP throughput with original microtime()

RBP TCP with transmission timer



RTC performance

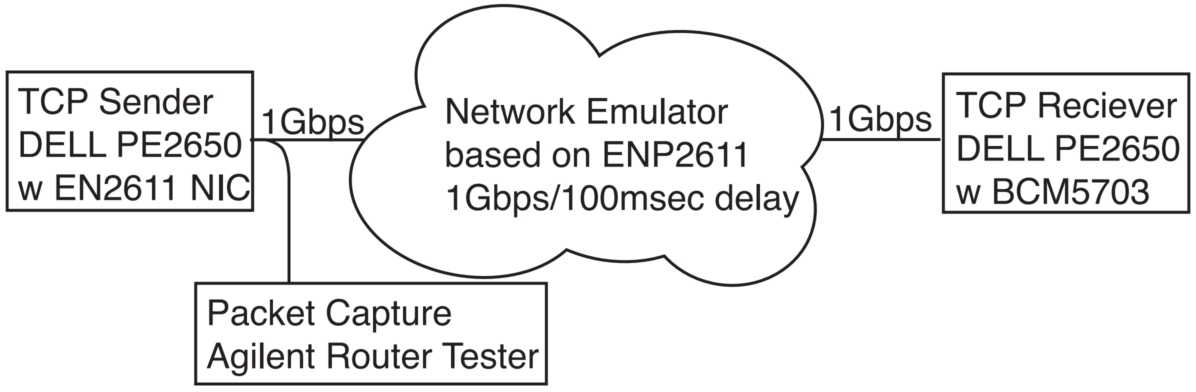
- 2 function is provided by original FreeBSD
    - microtime(): accesses RTC, higher accuracy result.
    - getmicrotime(): returns cached value that is updated each scheduler interrupt
- light microtime(): to correct getmicrotime() result with CPU TSC (Time Stamp Counter) on x86
    - TSC counts CPU clock cycle
    - Used for performance evaluation for program code.
    - x 40 faster than original microtime() with enough accuracy.

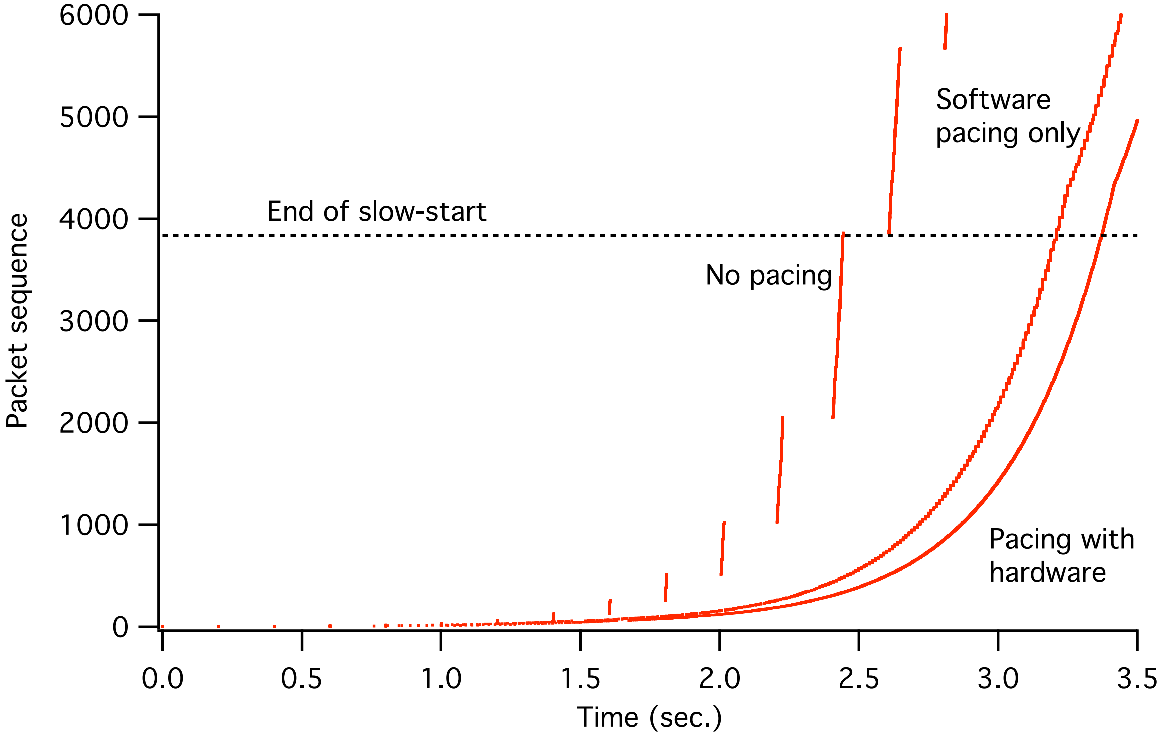
	elapsed TSC elapsed time	accuracy of result	packet throughput limit due to RTC overhead
microtime()	16,000 count 5 micro sec.	micro sec.	200K packets/sec. 2.4Gbps@1,500 Bytes/packet
getmicrotime()	89 count < 30 n sec.	1-10 m sec. (depend on scheduler)	330M packets/sec. 3Tbps@1,500 Bytes/packet
light microtime()	360 count 120 n sec.	micro sec.	83 M packets/sec. 1Tbps@1,500Bytes/packet

RBP TCP experiment

- 1Gbps/100 m sec. delay (200 m sec.RTT) between sender and receiver
  - no bottleneck between sender and receiver
  - no competitive flow

  - Agelent Router Tester captured packets on sender side.



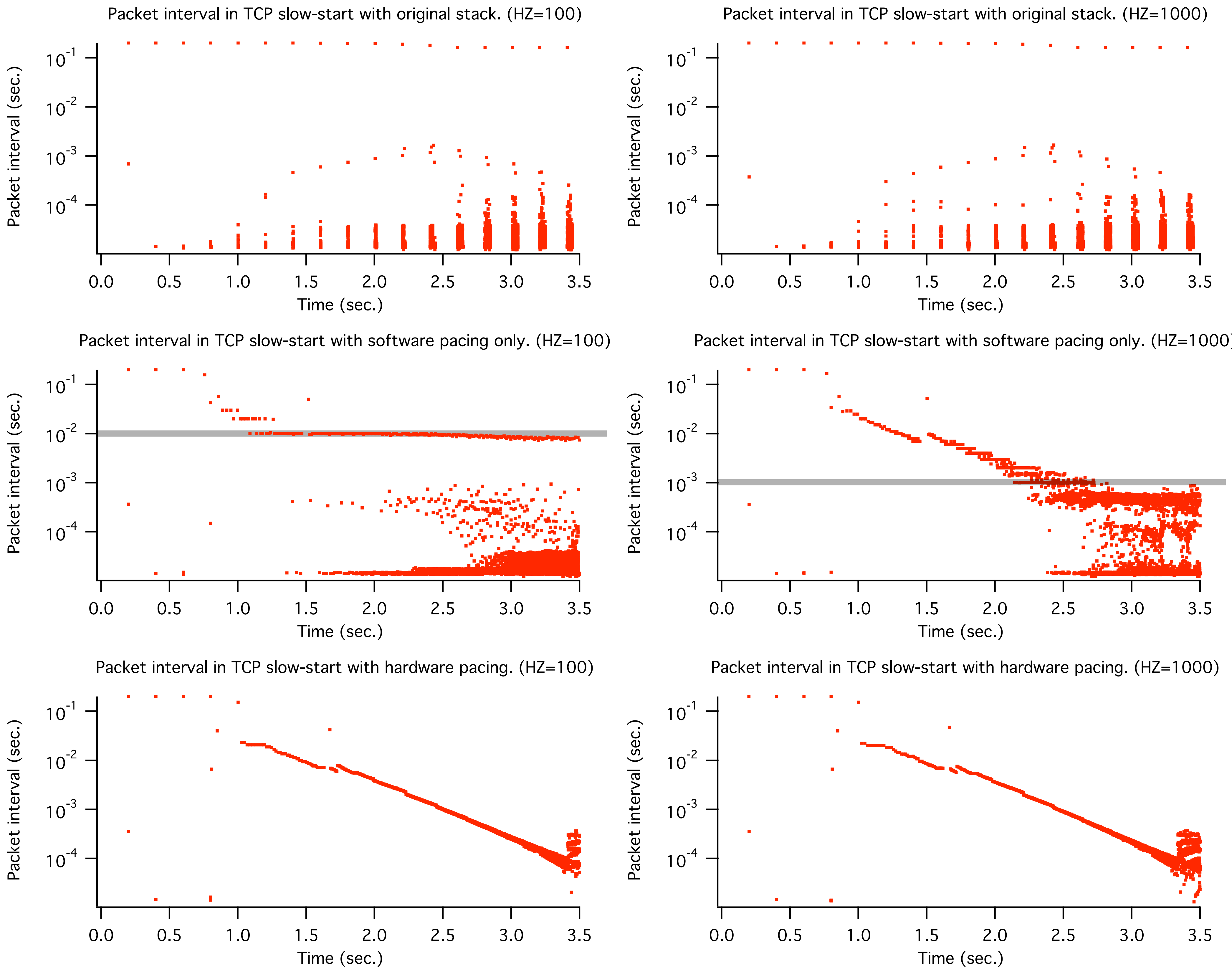


CPU tick:10 m sec. | CPU tick:1 m sec.

No RBP

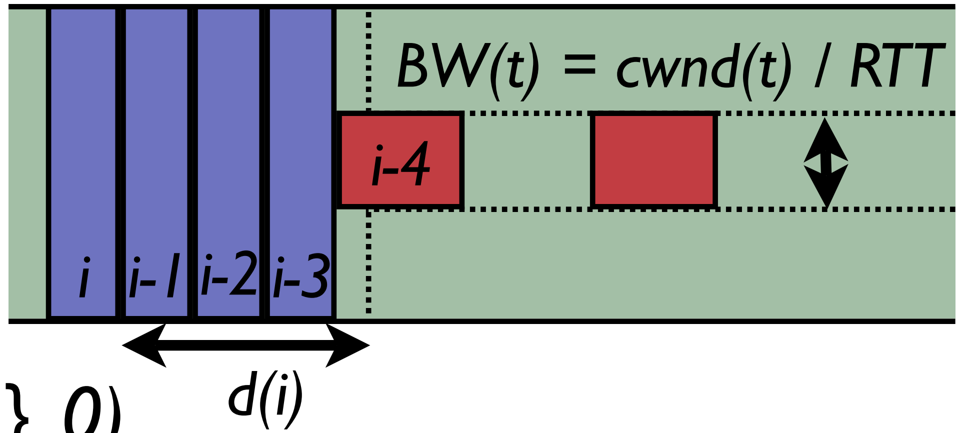
RBP with software only

RBP with Hardware



Buestiness index on slow-start

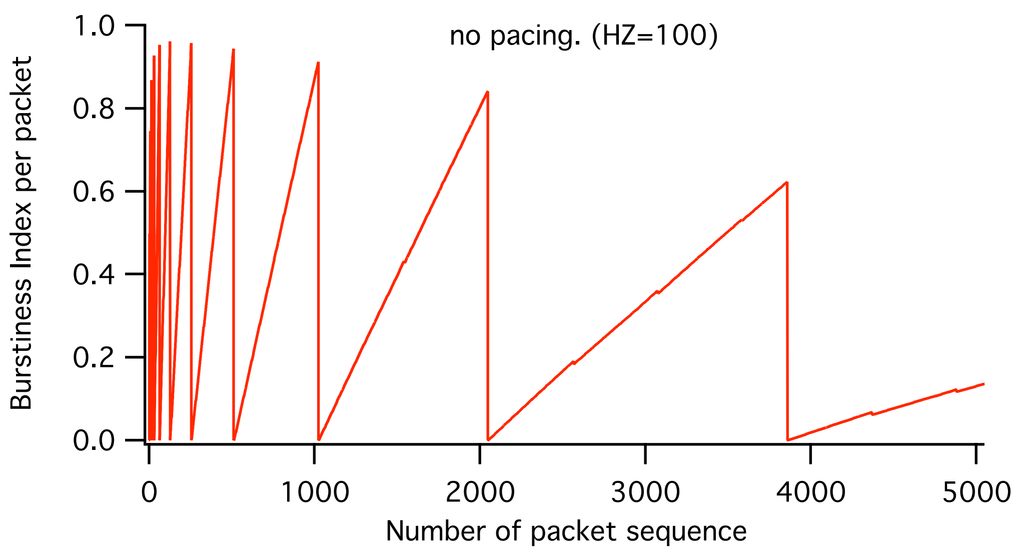
- Bandwidth *BW(t)* and serialization delay *sigma(t)* derived as:
  - $BW(t) = cwnd(t) / RTT$, $sigma(t) = RTT/cwnd(t)$
- *cwnd(t)* is assumed in exponential growth:
  - $cwnd(t) = 2^{(t / RTT)} \times MS$
- Packet delay *d(i)* at bottleneck:
  - $d(i) = max(d(i - 2) + sigma(t_i) - t_i - t_{i-1}, 0)$

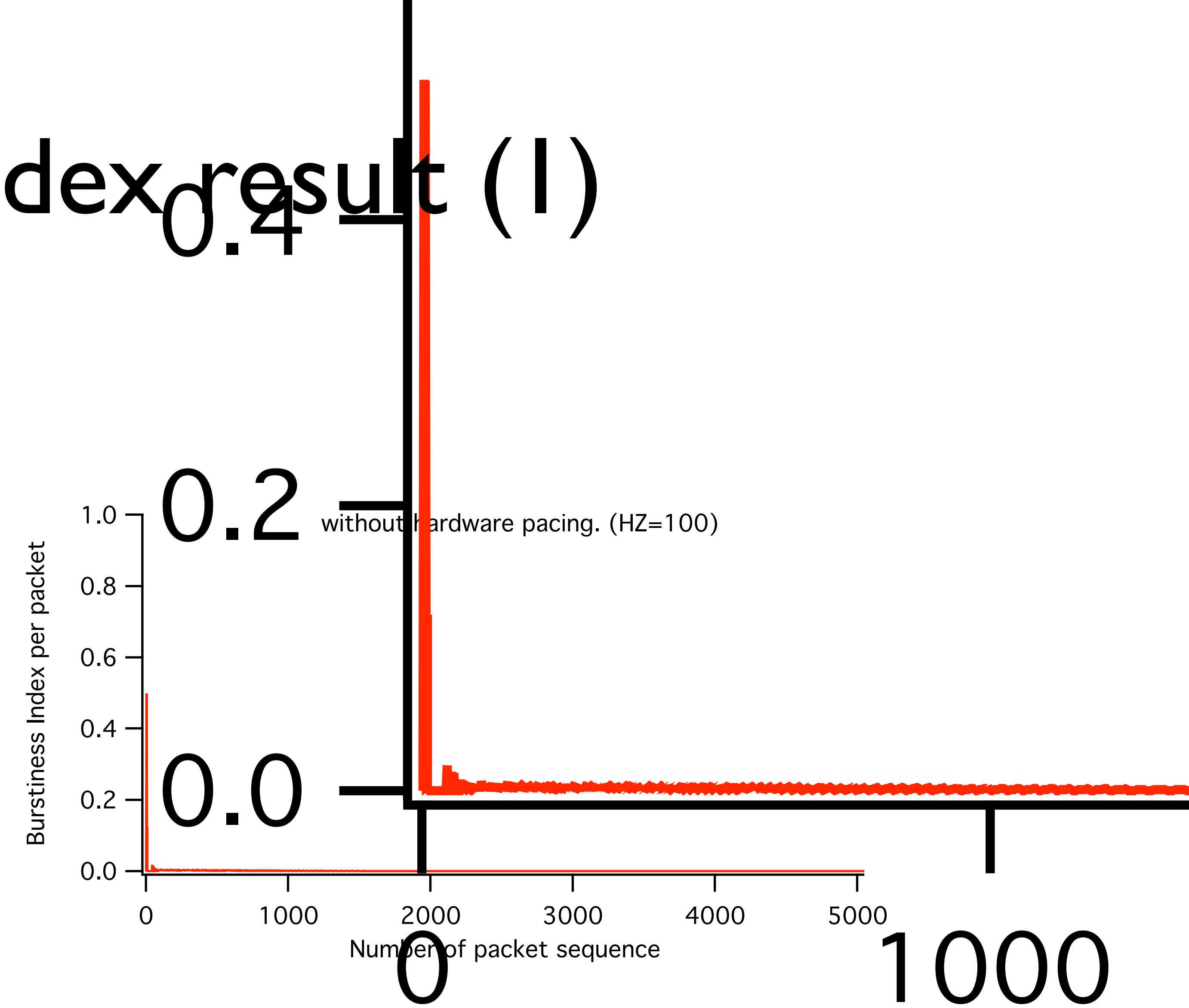


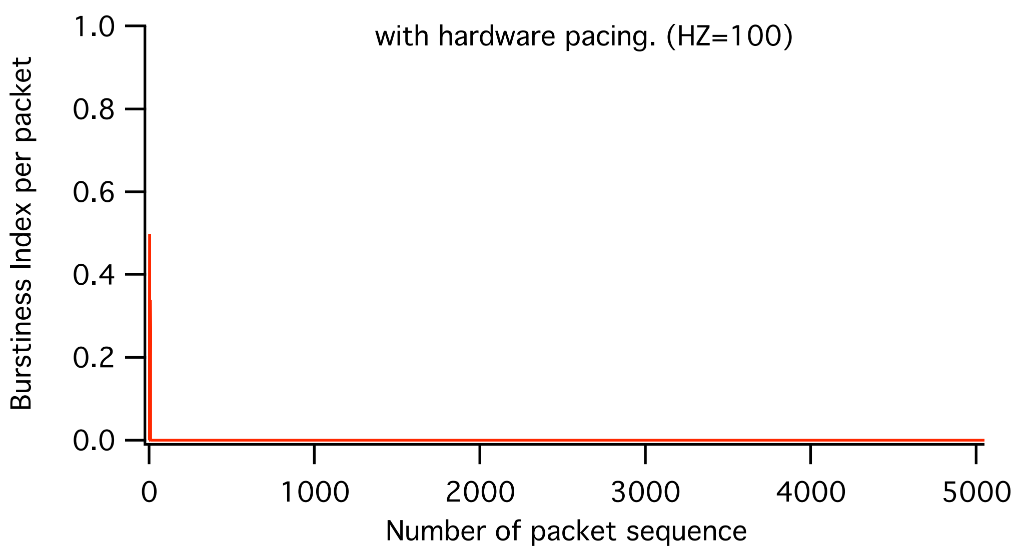
- Burstiness index b is derived normalizing with RTT:
  - $b(i) = d(i)/RTT$, $B = sum(b(i))$
- *b(i)* represents how much buffer is required at bottleneck by the stream itself.
  - If b(i) = 1: delay-bandwidth product size
  - If b(i) = 0: no buffer, complete RBP.

NiCT

Buestiness index result (1)







	10 msec.	1 msec.
RBP w H/W	1.5e-7	1.5e-7
RBP wo H/W	4.1e-4	5.0e-6
no RBP	0.31	0.31

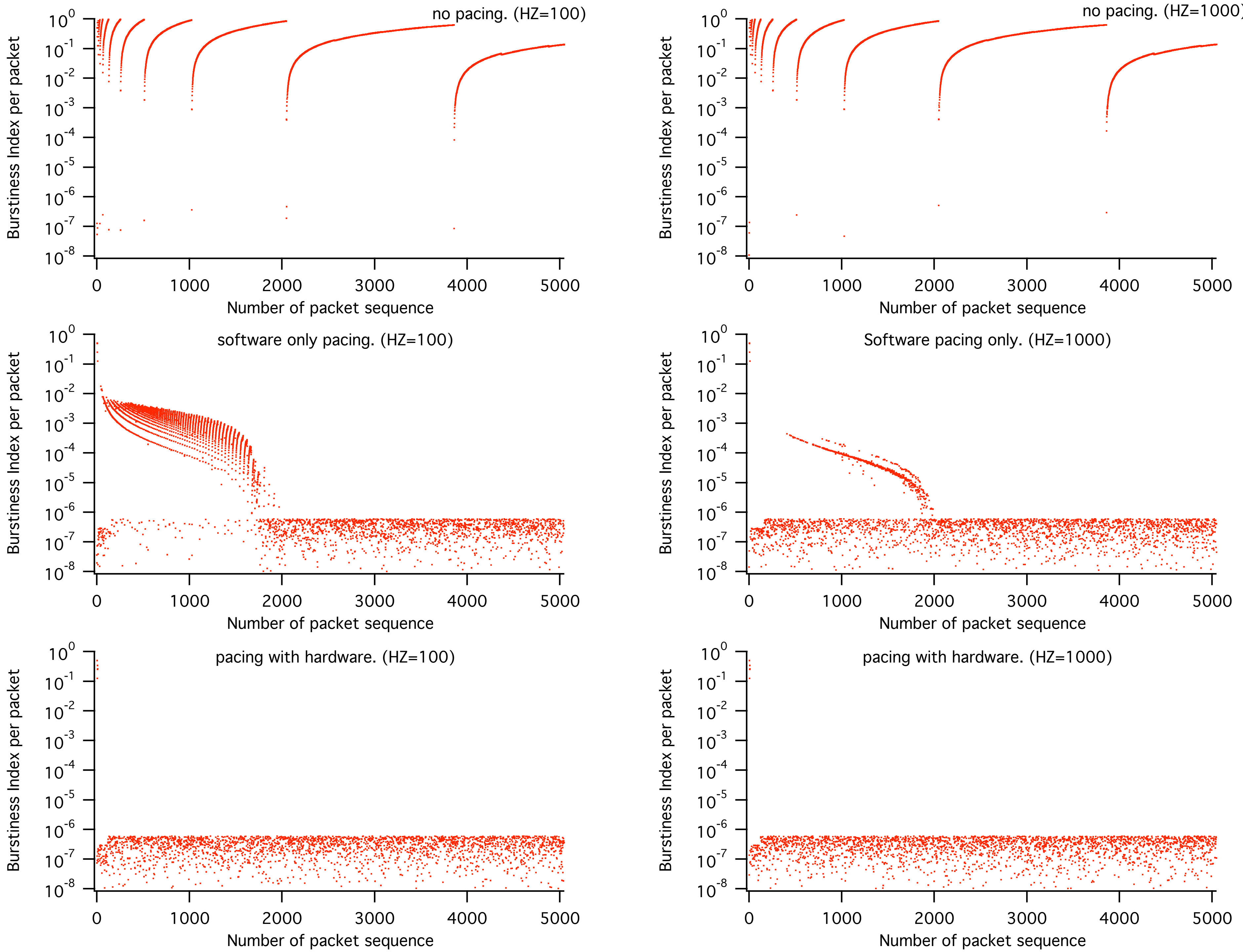
Buestiness index result (2)

CPU tick:10 m sec. | CPU tick:1 m sec.

No RBP

RBP with software only

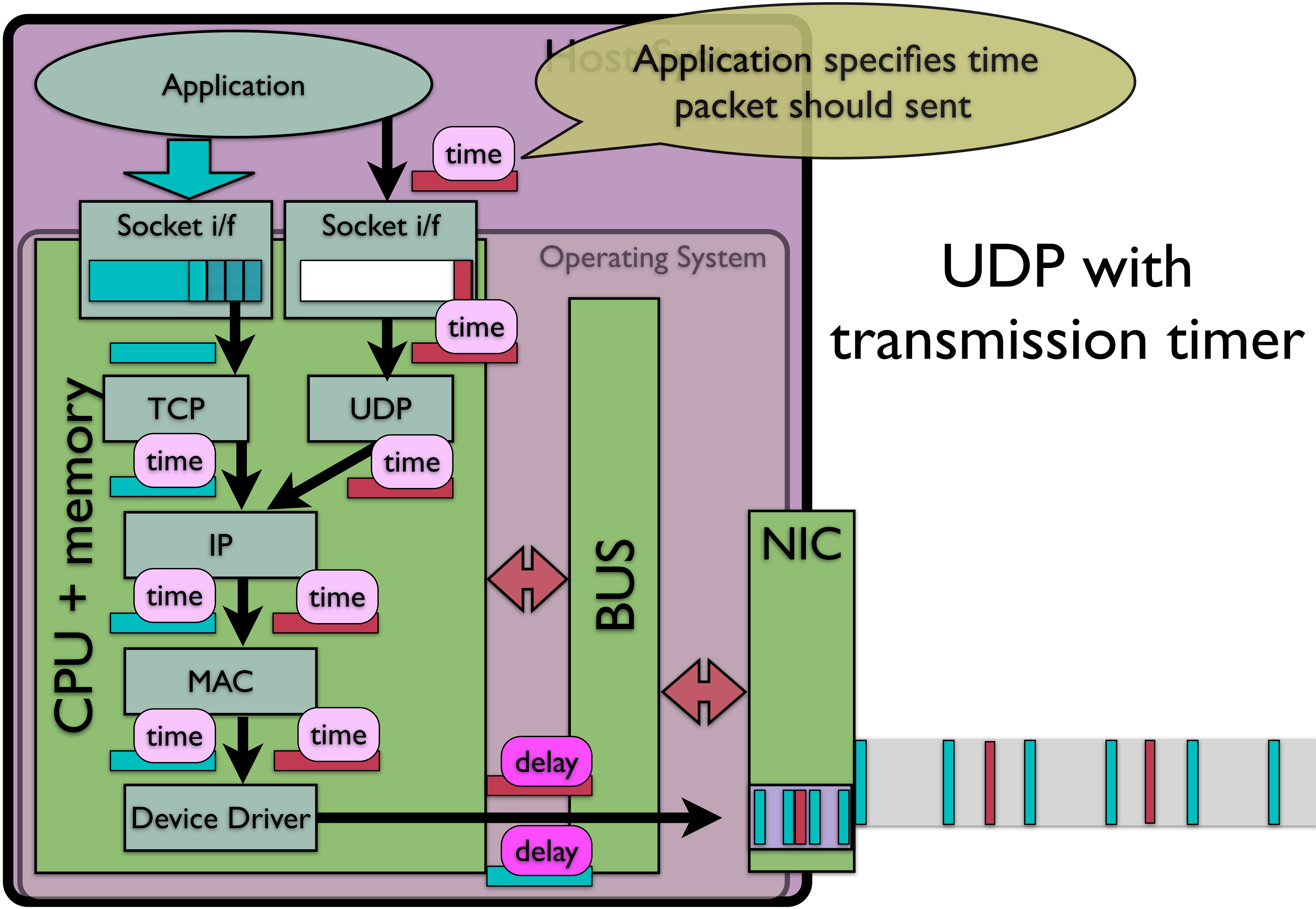
RBP with Hardware



Transmission Timer for UDP

- SO_TIMESTAMP is provided to obtain packet received time information.
  - struct timeval data accommodated as ancillary data with recvmsg() system call.
  - Receive only.
- To extend SO_TIMESTAMP at sending UDP datagram to specify the time that the application expects.

UDP with transmission timer



Conclusion

- Transmission timer approach for RBP
  - Framework
  - Implementation
  - Result

- Is our approach including interlayer interaction worse ?
- Does NIC vender interest this kind approach ?