PSPAT: Sofware Packet Scheduling at Hardware Speed

L. Rizzo, P. Valente, G. Lettieri, V. Maffione

Dipartimento di Ingegneria dell'Informazione
Università di Pisa

Work supported by Horizon2020 project SSICLOPS

NetV, 1/2/2017, Rennes

Outline

1. Problem
2. PSPAT
3. Performance
4. Conclusions

Problem and use case

- Network links way too fast
- NICs and operating systems are catching up
- VMs are catching up, too
- Demand for Network Function Virtualization

Cloud operators need robust, fast packet scheduling

- resource management
- isolation and protection

Software Packet scheduling not there yet

Why do we care about software schedulers

First block in the network path for VMs

- there are legitimate users with high PPS
- need to protect the virtual switch and the rest of the stack

Hardware does not always give perfect isolation

- the bus (PCIe) can be a bottleneck
- scheduling after the bottleneck is ineffective

Scheduling

Contract between client and resource manager

If YOU behave, I'll give you some service guarantee

Guarantees (rate allocation, loss, latency...):

- should be a binding promise by the system
- should not be affected by others' behaviour

We need

- algorithms with provable service guarantees
- know and robust behaviour under overload (practice)
- overload may cause failure to guarantee rate or latency

Terminology

Flow

- collection of traffic managed as a single entity by the SA
- number can be large

Scheduling Algorithm (SA)

- just the algorithm that decides which flow to serve next
- different goals (priority, latency, fixed or weighted rate allocation)
- can reason about complexity and guarantee

Packet Scheduler

- the whole system: Queues + Scheduling Algorithm + I/O
- operating conditions and performance are an issue

Requirements and state of the art

Requirements

- request rates up to 1–10 Mpps
- latency $< 10\,\mu s$

Theory gives us several Scheduling Algorithms Tradeoffs between complexity and guarantees (T-WFI)

- DRR (deficit round robin): 20 ns/decision, $O(N)$ delay
- WF2Q+ (Weighted Fair Queueing): $O(\log N)$ time, $O(1)$ delay
- QFQ/QFQ+ (Quick Fair Queueing): 40–50 ns/decision, $O(1)$ delay

Practice gives us some promising tools

- fast CPU, buses, multiport NICs
- fast I/O frameworks

Traditional Software Packet Scheduler

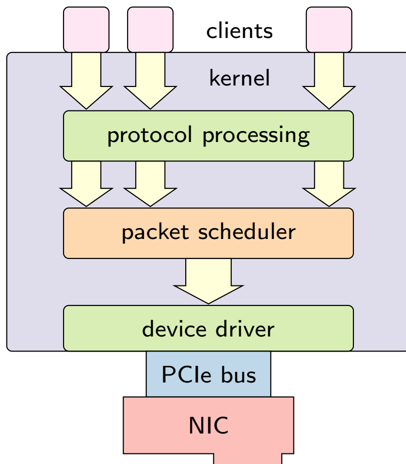
PROS

- no hardware dependencies
- large choice of algorithms

CONS

- heavy lock contention in accessing the scheduler
- under congestion I/O becomes serialized
- scalability can be problematic

TC delivers 2 Mpps, decreasing with number of clients



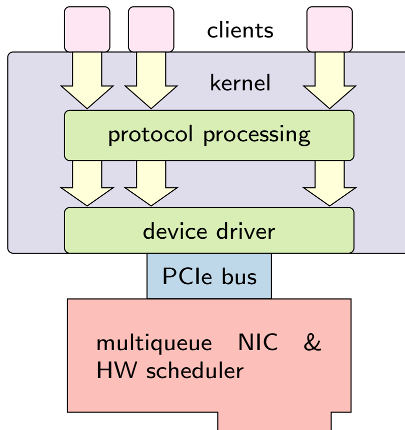
Hardware Packet Schedulers

PROS

- fully parallel down to the NIC
- reduced system load due to HW offloading

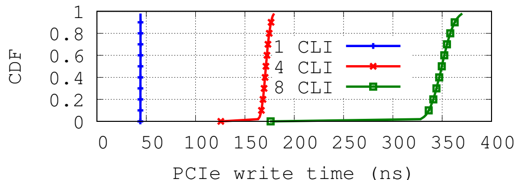
CONS

- limited choice of algorithms
- the bus is still a point of contention.



PCIe access issues

- PCIe arbitration is round robin, *not programmable*
- PCIe service rate is limited by the NIC
- PCIe bus can saturate as well



Latency distributions in µs on I7

CLI	Notes	Percentile min	10	50	90	99
1	HW	5.7	5.8	6.0	6.1	6.4
1	TC	5.5	5.7	5.9	6.1	6.6
1	PSPAT	6.3	6.8	7.2	7.7	8.2
5	HW (PCIe congestion)	9.8	117.0	125.0	137.0	152.0
5	TC @ 10G .812 Mpps	6.6	8.5	12.6	16.6	18.6
5	PSPAT @ 10G .823 Mpps	6.4	7.3	9.0	11.1	12.2

Dilemma

SW flexible but slow, HW not as good as we would like

1. Denial: we don't need fast schedulers
   - what about NFV?
2. Faith: hardware will get better
   - what about existing hardware?
3. Various approximate solutions
   - trivial schedulers (FIFO, DRR: fast but poor delay guarantees)
   - active queue management (RED, CODEL: rely on everyone behaving)
   - bounded number of queues: rely on quiet neighbours

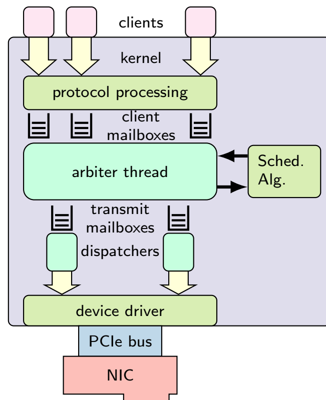
PSPAT: Packet Scheduling with PArallel Transfers

Decouple scheduling and transmission

- a dedicated arbiter thread runs the SA (sequential)
- traffic is released at link rate to the device driver
- possibly one or more threads perform transmission in parallel

Results

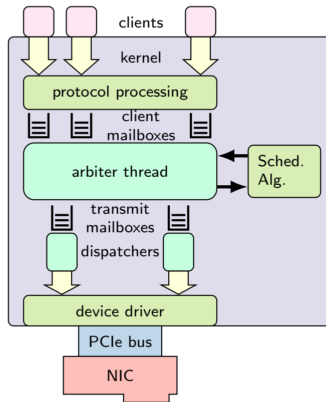
- reduced contention, increased parallelism
- large speedup compared to TC
- the architecture permits a worst case analysis



PSPAT components

- M Clients and Client Mailboxes $CM[x]$
- C cores on a shared memory system
- C Client Lists $CL[c]$, clients recently active on core c
- 1 ARBITER
- N flows
- $1–T$ Transmit queues on the NIC
- $0–T$ Transmit Mailboxes and Dispatcher

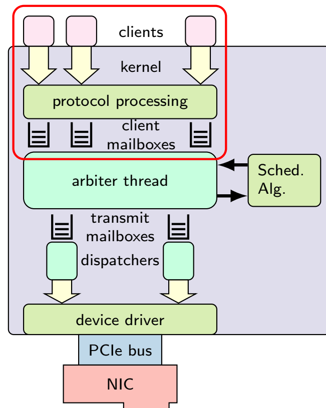
N and M can be very large. T can be kept small as needed.



CLIENT operation (1/2)

Client x on core c wants to transmit packet p:

- insert p into $CM[x]$
- if $(CL[c].tail \; != x)$ append x to $CL[c]$



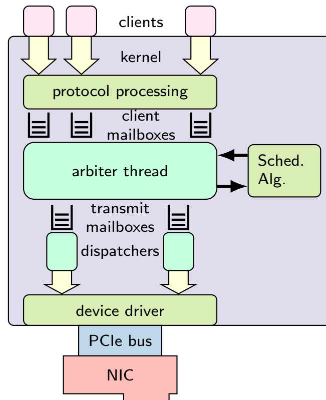
CLIENT operation (2/2)

PROS

- no contention accessing the lock-free mailbox
- backpressure through mailbox-full indication
- some work offloaded to arbiter and dispatcher

CONS

- possible additional latency
- possibly, reduced backpressure (similar to CODEL)



MAILBOX

Memory-based interactions can be slow

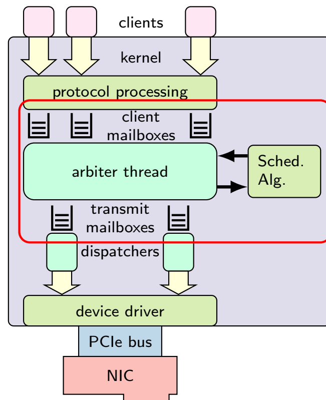
	HT-HT	Core-Core	SKT-SKT
Read stall	10-15 ns	50 ns	130-220 ns
Write stall	–	15 ns	100 ns
Updates per second	75 M	20 M	5 M
Round trip latency	30 ns	130 ns	480 ns

- avoid passing queue pointers, as in FastForward (saves a barrier)
- use slot state (NULL or not) to indicate new packets
- avoid W-W conflicts by releasing slots lazily
- reduce R-W conflicts by rate limiting memory accesses

ARBITER operation (1/2)

Continuously runs a three-phase task

1. grab packets from active $CM[x]$ ($\approx 1CM$ per core), `enqueue()` packets to the Scheduling Algorithm; trim client lists $CL[c]$
2. `dequeue()` packets at link rate (leaky bucket) to TM
3. notify Dispatchers for new work



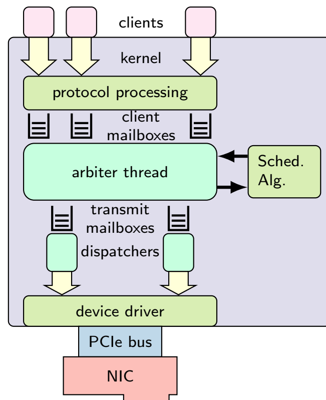
ARBITER operation (2/2)

PROS

- arbitrary choice of Scheduling Algorithm
- arbitrary split/merge of traffic
- no lock contention

CONS

- needs to scan all mailboxes — scale?



ARBITER: Limit number of mailboxes to scan

Only need to look at mailboxes that may have traffic:

- only clients which ran on a core and sent packets since the previous scan
- Client Lists keep track of those clients
- scans are very frequent (every few μs)

Client Lists rarely have more than 1 entry.

ARBITER: scan speed

- inactive *CL*s and *CM*s likely in L1, so only 2–4 ns each
- active entries may cause a read stall, rate limit access to keep stalls within 10%
- plus of course `skb` access and `enqueue()` cost
- try to keep scan within 2–5 μs

Some self correcting mechanisms if a round is slow:

- caching and batching more effective
- mailboxes fill up, stopping the high speed clients

DISPATCHER

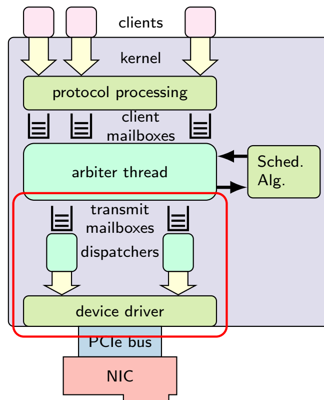
- grab packets from Transmit Mailbox, run device driver

PROS

- number can be configured as needed
- operate at controlled rate, never congested
- can be implemented in the arbiter if using a fast backend

CONS

- potentially, extra latency



PSPAT implementation

Two versions

- in-kernel, for complete compatibility with TC:
  - intercept traffic in `__dev_queue_xmit()`,
  - deliver to `dev_hard_start_xmit()`
  - reuses Linux QDISC code
- userspace, for fast prototyping and optimized performance
  - supports userspace networking (netmap, DPDK...)
  - can use fast scheduling code from dummynet

Performance analysis

Metrics:

- throughput and latency

Platforms:

- i7 with 40G NIC and Linux 4.7 (in-kernel PSPAT)
- dual Xeon E5-2640 (userspace)

Sources (one per core, pinned):

- UDP sockets (not very fast)
- pkt-gen (the netmap version), very fast
- Linux pktgen bypasses __dev_queue_xmit()

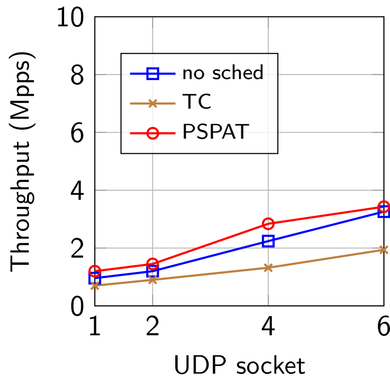
Packet schedulers:

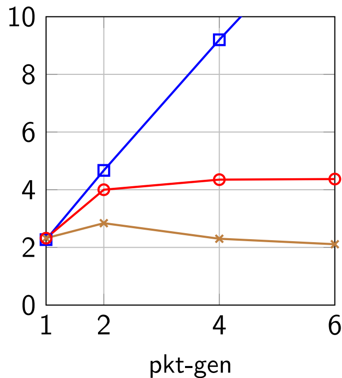
- none (HW), TC, PSPAT

Throughput measurements

- Clients send as fast as possible
- variable number of clients
- schedulers use QFQ (DRR is marginally faster)
- TC and PSPAT rates higher than scheduler's capacity
- measurements in PPS as that is the relevant metric

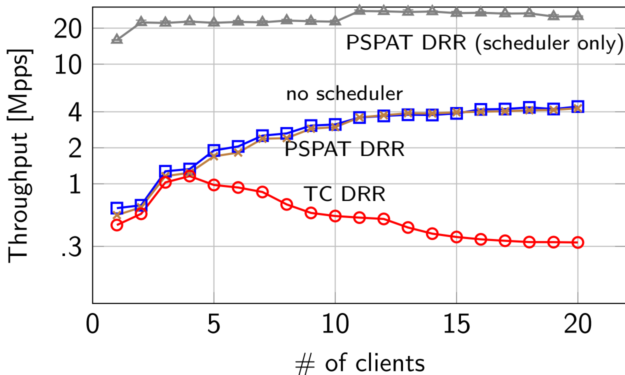
Throughput with regular UDP (i7)





- $2\times$ speedup (limited by the qdisc code)

Throughtput for userspace PSPAT (Xeon)



Rate allocation

Not shown in the graphs: *TC fails to meet rate allocation!*

- every `enqueue()` followed by `dequeue()`
- scheduler slower than link means there is never any queueing
- allocation just matches request rate

PSPAT addresses this

- first grab all requests, then run dequeue()
- queues build up even when the scheduler is slow

One way latency measurements

Experiments with different link rates and number of clients

- one client has weight=100, sends at half the reserved bandwidth
- other clients have weight=1, send as fast as possible

Theory says latency is proportional to MSS/RATE

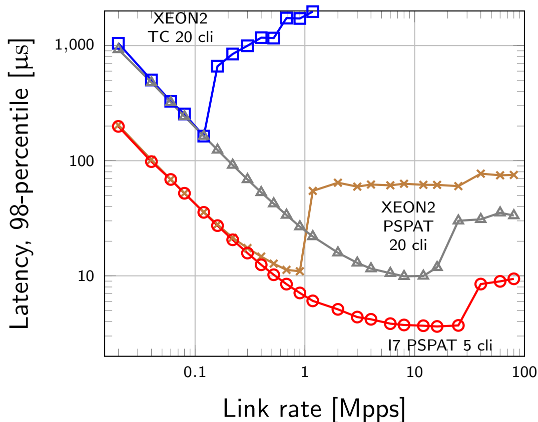
One way latency measurements

Latency distributions in μs on I7

CLI	Notes	Percentile min	10	50	90	99
1	HW	5.7	5.8	6.0	6.1	6.4
1	TC	5.5	5.7	5.9	6.1	6.6
1	PSPAT	6.3	6.8	7.2	7.7	8.2
5	HW (PCIe congestion)	9.8	117.0	125.0	137.0	152.0
5	TC @ 10G .812 Mpps	6.6	8.5	12.6	16.6	18.6
5	PSPAT @ 10G .823 Mpps	6.4	7.3	9.0	11.1	12.2

- No big surprises for PSPAT: a couple of extra μs due to rate-limited scans and handoffs
- Note the huge effect of congestion on the PCIe bus

Latency versus rate



Conclusions

Paper:
http://info.iet.unipi.it/~luigi/papers/20160921-pspat.pdf.
Code:
https://github.com/giuseppelettieri/linux-pspat.

Thank you!