ELI: Bare-Metal Performance for I/O Virtualization

Abel Gordon[×] Nadav Amit[¤] Nadav Har'El[×]
Muli Ben-Yehuda[×,¤] Alex Landau[×] Assaf Schuster[¤] Dan Tsafrir[¤]

[×] IBM Research – Haifa
[¤] Technion – Israel Institute of Technology

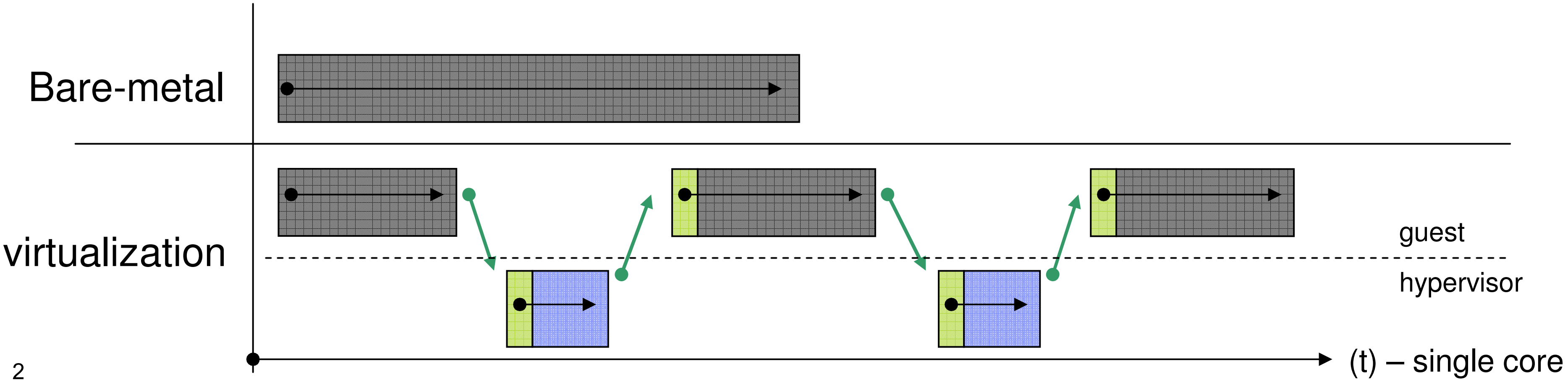
Partially supported by the European Community's Seventh Framework Programme([FP7/2001-2013]) under grant agreements number 248615 (IOLanes) and 248647 (ENCORE)

Background and Motivation

- Virtualization already is an integral part of our systems

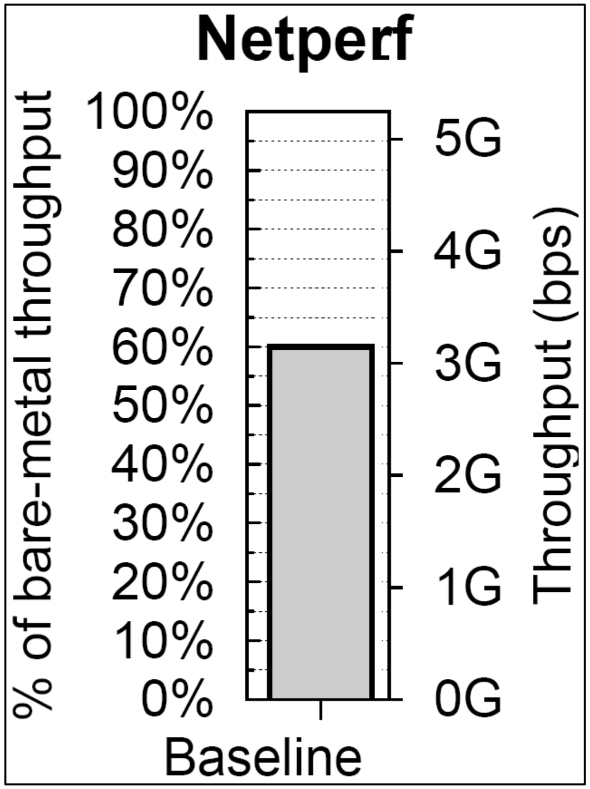
- Virtualization overhead is high for a common subset of workloads, in particular I/O intensive workloads

- Overhead causes:
  - ●→ – Context switch cost (e.g. switches between the hypervisor and the guests)
  - 🟩 – Indirect cost (e.g. CPU cache pollution)
  - 🟦 – Handling cost (e.g. handling external interrupts)



I/O Intensive Workloads

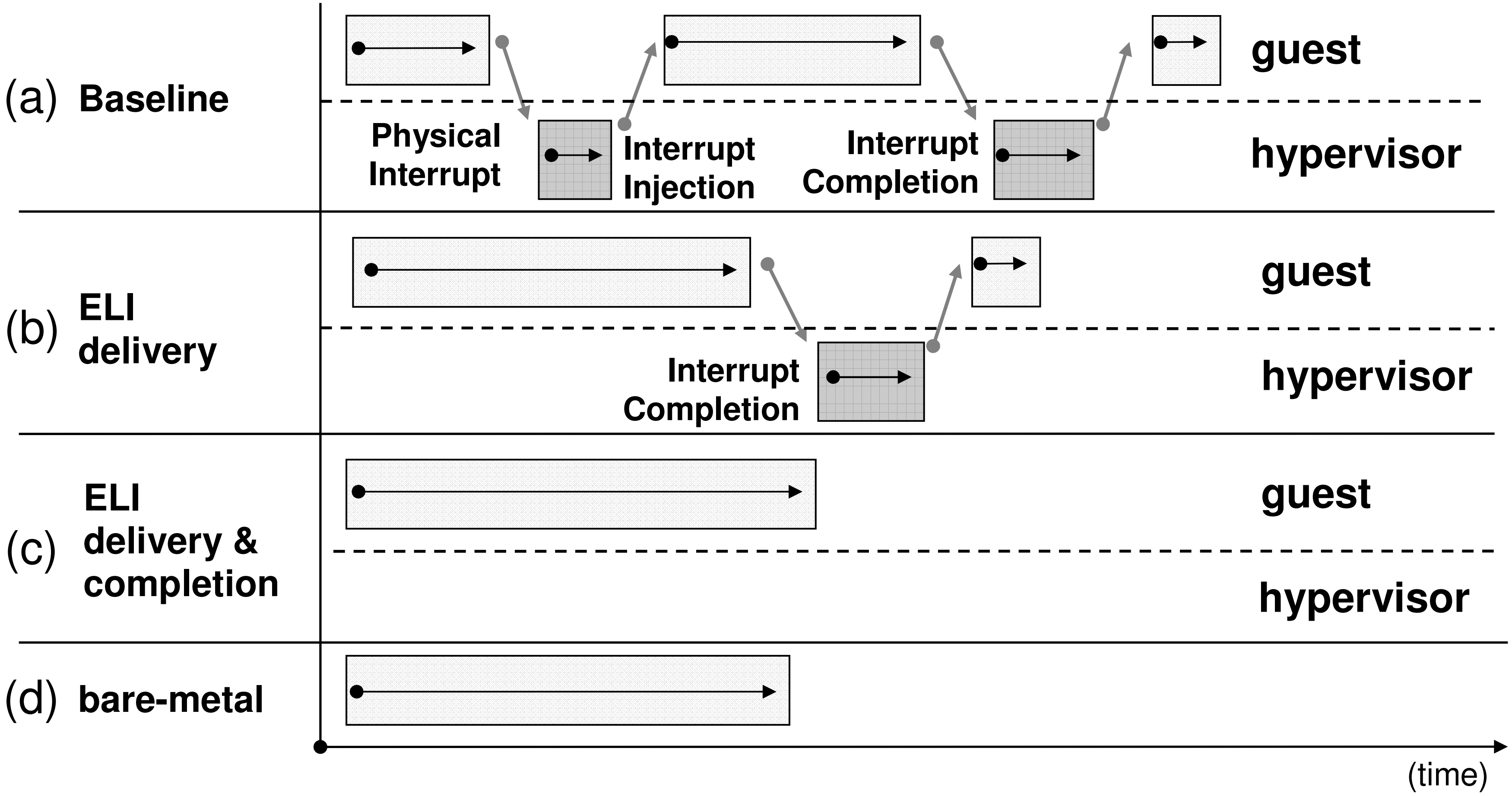
- Best performing model: Device Assignment (SR-IOV devices)
  - – The guest has direct access to a dedicated physical device (DMA and MMIO)
  - – No hypervisor intervention …**except for interrupt handling**
- Overhead still high compared to bare-metal (non-virtual) [Adams06, Ben-Yehuda10, Landau11]
  - – Switches to the hypervisor due to external interrupts arriving from the device
  - – Switches to the hypervisor due to interrupt completions signaled by the guest
- Overhead is visible as [Liu10, Dong10]
  - – Lower throughput (when the CPU is saturated, usually for small messages)
  - – Higher CPU consumption (when line rate is attained, usually for big messages)
  - – Higher latency



ELI: ExitLess Interrupts

●→ guest/host context switches (exits and entries)

▨ handling cost (handling external interrupts and interrupt completions)



(a) Baseline

guest

Physical Interrupt

Interrupt Injection

Interrupt Completion

hypervisor

(b) ELI delivery

guest

Interrupt Completion

hypervisor

(c) ELI delivery & completion

guest

hypervisor

(d) bare-metal

(time)

Related Work

- Polling
  - Disables interrupts and polls the device for new events
  - Adds latency, waste cycles, consumes power
- Hybrid [Dovrolis01, Mogul96, Itzkovitz99]
  - Dynamically switches between interrupts and polling
  - Default in Linux (NAPI) [Salim01]
  - Hard to predict future interrupt rate
- Interrupt Coalescing [Zec02, Salah07, Ahmad11]
  - Limits interrupt rate (sends only one interrupt per several events)
  - Adds latency [Larsen09, Rumble11], might burst TCP traffic [Zec02], complex to configure and change dynamically [Ahmad11,Salah08], adds variability

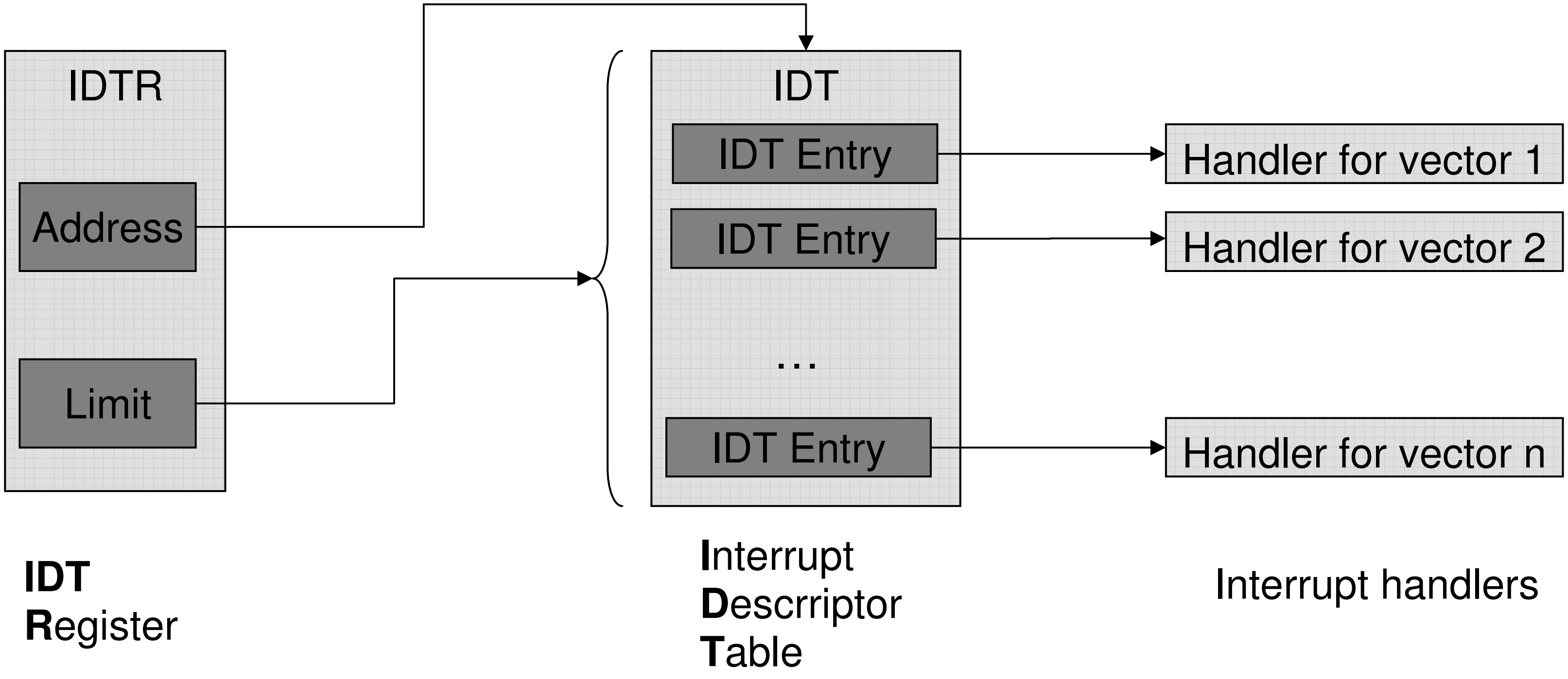
ELI is **complementary** to these approaches:

(1) **Removes the virtualization overhead** caused by the costly exits and entries during interrupt handling

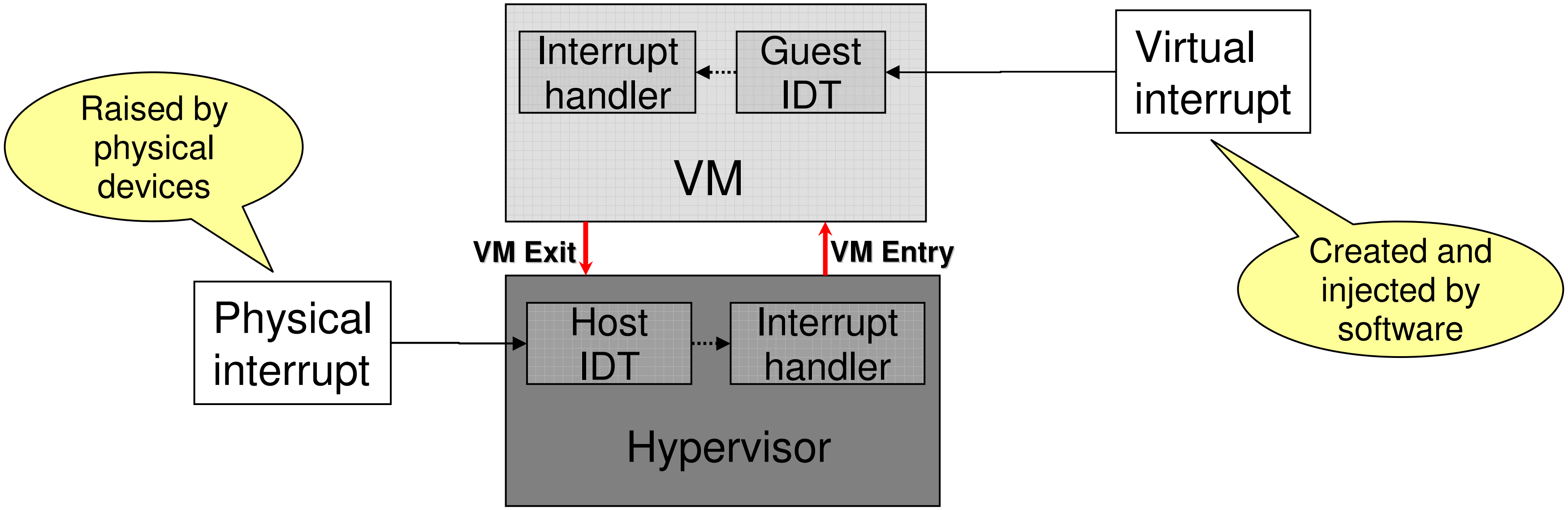
(2) Lets the **guest control directly** the interrupt rate and latency

x86 Interrupt Handling

- Interrupts are asynchronous events generated by external entities such as I/O devices
- x86 CPUs use interrupts to notify the system software about incoming events
- The CPU temporarily stops the currently executing code and jumps to a pre-specified interrupt handler
- Hardware and software identifies interrupts using vector numbers.



x86 Interrupt Handling in Virtual Environments

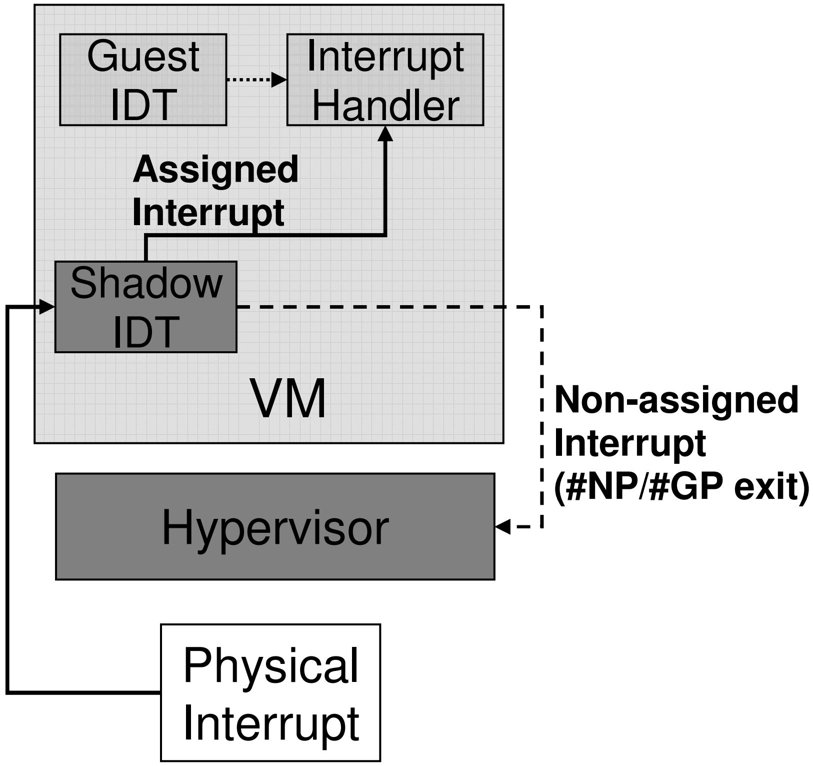


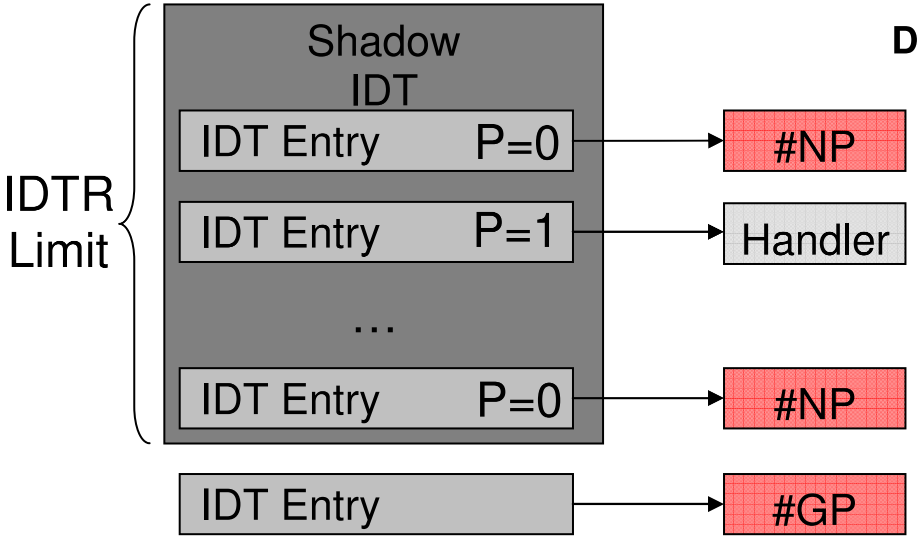
- Two IDTs
  - <u>Guest IDT</u>: handles virtual interrupts created by the “virtual” hardware
  - <u>Host IDT</u>: handles physical interrupts raised by the “physical” hardware

- If a physical interrupt arrives while the guest is running, the CPU forces a transition to the hypervisor context (VM Exit)
  - Required for correct emulation and isolation

ELI: ExitLess Interrupts - Delivery

- Allow interrupt delivery directly to the guest
  - Configure the hardware to deliver all interrupts to the guest (CPU only supports all or nothing mode)
  - Control which interrupts should be handled by the guest and which interrupts should be handled by the host using a shadow IDT





ELI: ExitLess Interrupts - Completion

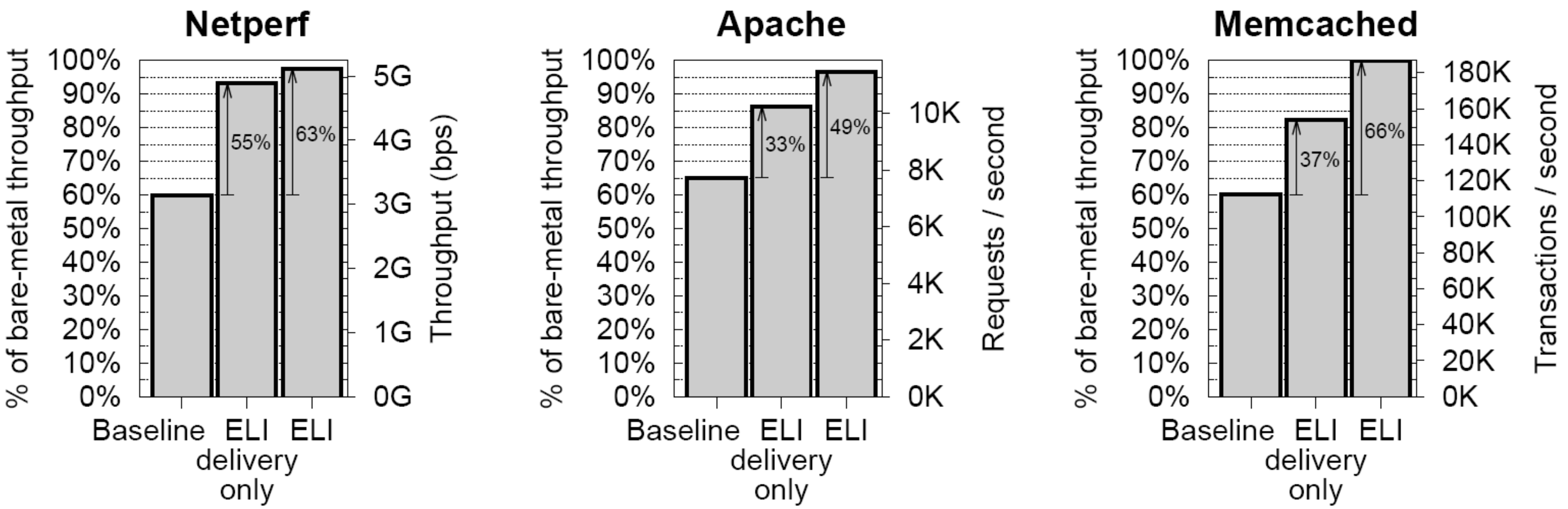
- The guest OS signals interrupt completions writing to the Local Advance Programmable Interrupt Controller (LAPIC) End-of-Interrupt (EOI) register

- Old LAPIC interface
  - The guest accesses the LAPIC registers using regular load/stores to a pre-specified memory page
  - The hypervisor traps accesses to the LAPIC page (almost all registers)

- New LAPIC interface (x2APIC)
  - The guest accesses LAPIC registers using Machine Specific Registers (MSRs)
  - The hypervisor traps accesses to MSRs (LAPIC registers) using hardware virtualization MSR bitmap capability

- ExitLess Completion
  - Requires x2APIC
  - ELI gives direct access only to the EOI register

Evaluation



- Throughput scaled so 100% means bare-metal throughput
- Throughput gains over baseline device assignment are noted inside the bars
- CPU is saturated in the 3 benchmarks

Parameters:

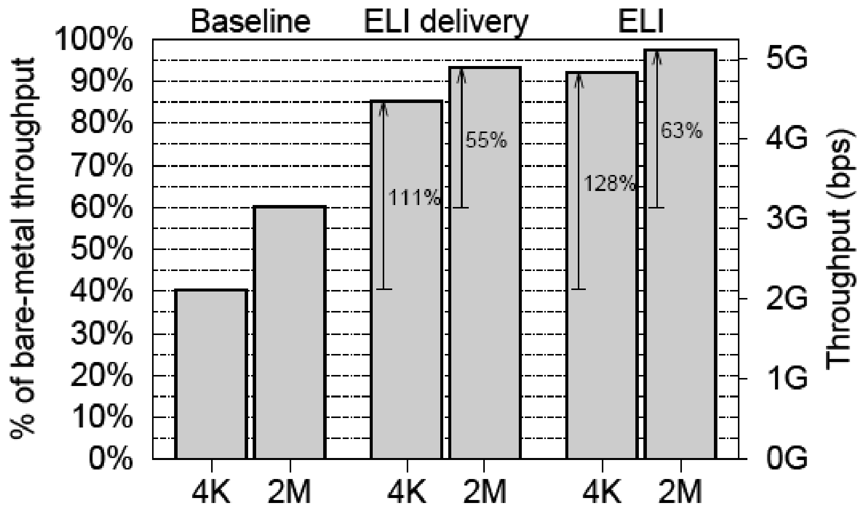
- KVM uses EPT + Huge Pages (host only)
- Netperf: 256B messages
- Apache: stressed with apachebench (4 threads requesting 4KB static pages)
- Memcached: stressed with memslap (4 threads / 64 concurrent requests, key size = 64B, value size = 1024B, get/set ratio = 9:1)
- x2APIC behavior emulated on a non-x2APIC hardware

Evaluation

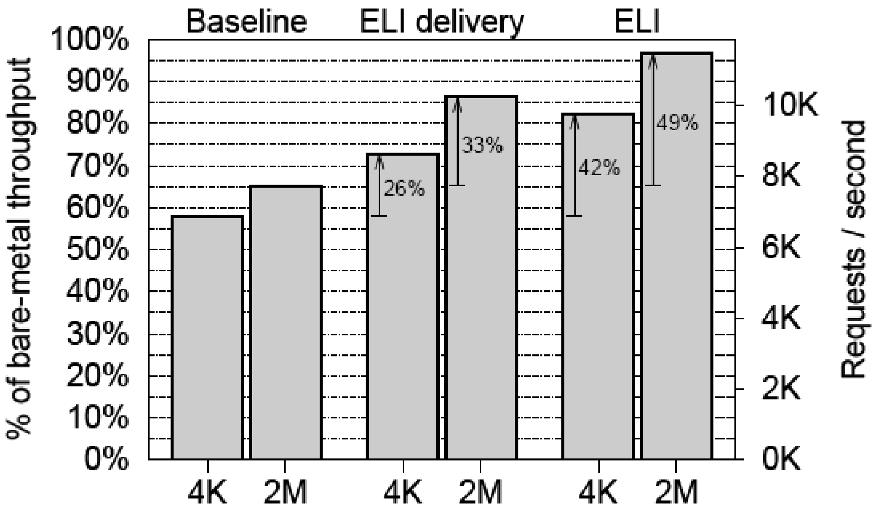
	Baseline	ELI Delivery	ELI
NetPerf			
Exits/s	102K	44K	0.8K
Time in guest	69%	94%	99%
% of bare-metal throughput	60%	92%	98%
Apache			
Exits/s	91K	64K	1.1K
Time in guest	67%	89%	98%
% of bare-metal throughput	65%	83%	97%
Memcached			
Exits/s	123K	123K	1K
Time in guest	60%	83%	98%
% of bare-metal throughput	60%	85%	100%

ELI removed most of the exits and almost achieved bare-metal performance!

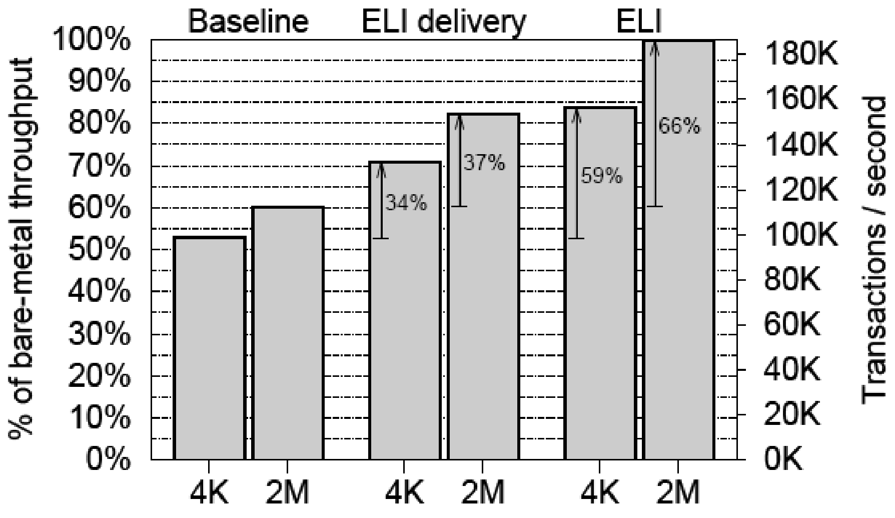
Huge Pages



Netperf



Apache

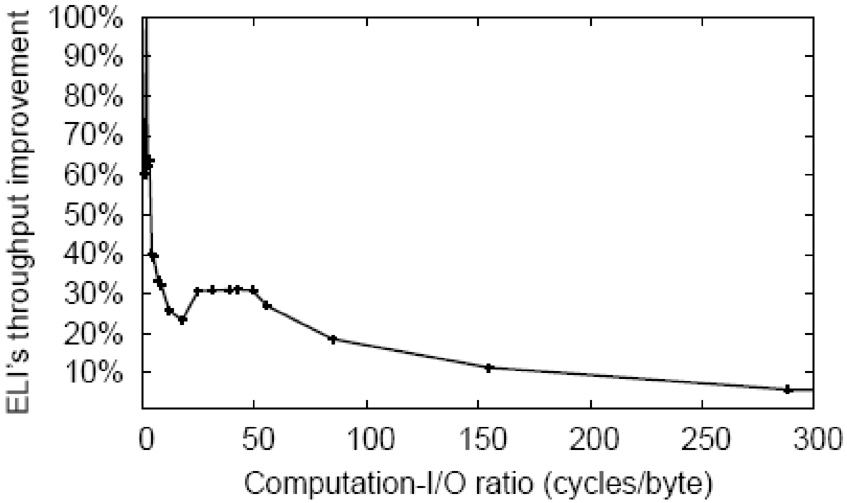


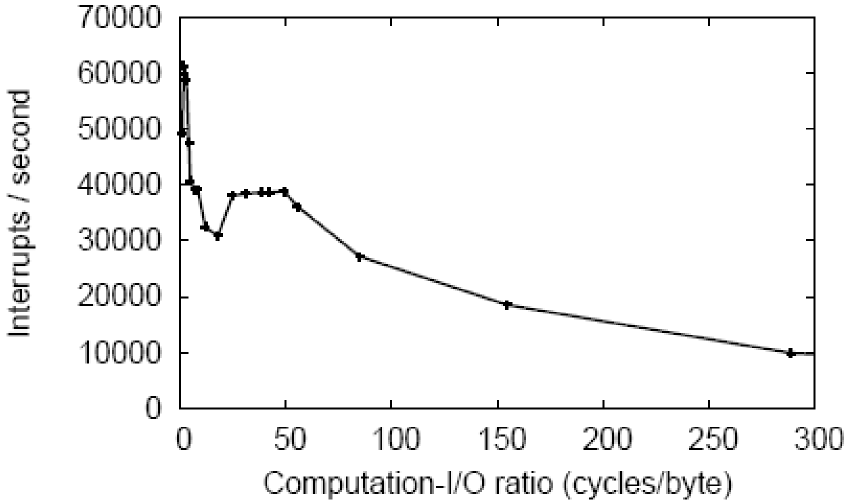
Memcached

(1) ELI significantly improved performance even without huge pages

(2) Huge pages are required to achieve bare-metal performance

Computation vs. I/O ratio (modified netperf)

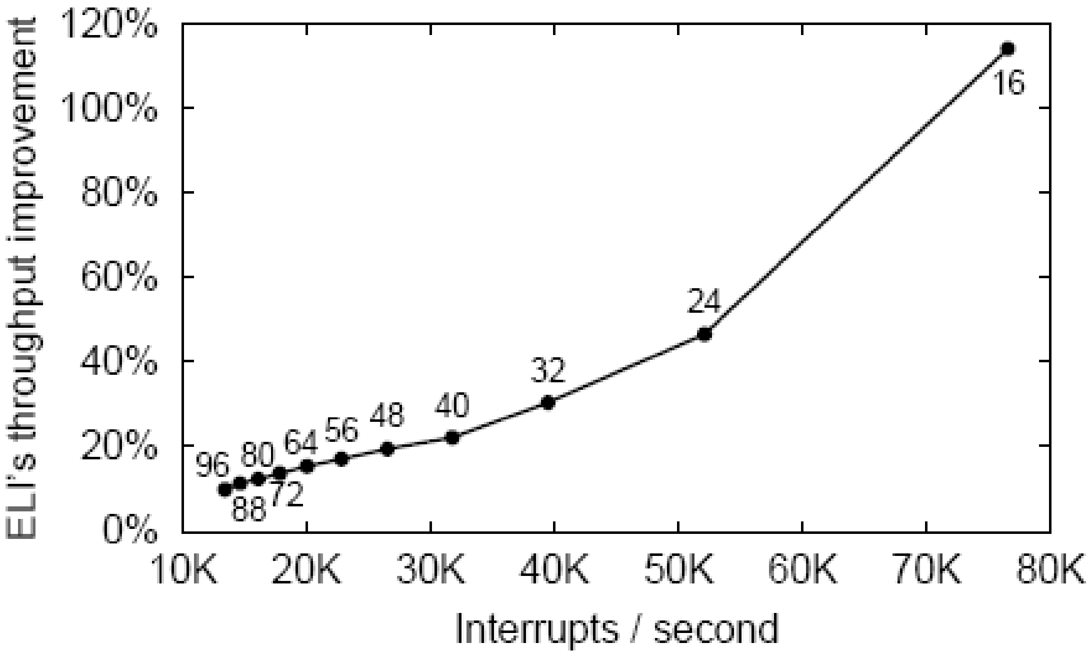




cycles/byte = CPU frequency (2.93GHz) / throughput

ELI's improvement remains high even for 50Mbps (60 cycles/byte) because NAPI and the NIC's adaptive coalescing mechanism limit the interrupt rate (interrupt rate is not always proportional to the throughput)

Interrupt Coalescing (netperf)



Even using maximum coalescing supported by the NIC (96μs), ELI provides 10% performance improvement

Latency

Configuration	Avg. Latency	% of bare-metal
Baseline	36.14μs	129%
ELI delivery-only	30.10μs	108%
ELI	28.51μs	102%
Bare-metal	27.93μs	100%

Netperf UDP Request Response

ELI substantially reduces the time it takes to deliver interrupts to the guest, critical for latency-sensitive workloads.

Implementation

- Locating the shadow IDT for unmodified guests
  - Shadow IDT must be mapped into the guest address space
  - Use PCI BARs (MMIO regions) to force the guest OS (Linux & Windows) to map and (keep mapped) additional memory pages
  - Write-protect the shadow IDT
- Injecting virtual interrupts
  - Use the original Guest IDT to inject a virtual interrupt
- Nested interrupts (a higher vector can interrupt a lower vector)
  - Check if a physical interrupt is being handled by the guest before injecting a virtual interrupt

Security, protection and isolation

- Threat: malicious guests might try consume interrupts, keep interrupts disabled or signal invalid completions

- ELI defends itself against malicious guest using multiple mechanisms:
  - Hardware Virtualization preemption timer to force exits (instead of relying on timer interrupts)
  - EOIs while no interrupt is being handled do not affect the system
  - Periodically check shadow IDT mappings
  - Protect critical interrupts
    - Deliver to a non-ELI core
    - Send spurious Interrupts (to re-create a possible lost interrupt)
    - Redirect as NMI (NMIs can be configured to force an exit unconditionally)
    - Use IDTR limit (reserve highest vectors for critical host interrupts)

Future Work

- Reduce frequency of exits caused by para-virtual I/O devices
  - Use ELI to send notifications from the host to the guest (running on a different cores) without forcing an exit
- Reduce frequency of exits caused by non-assigned interrupts
  - Shadow the interrupt handlers and batch/delay interrupts (host interrupts or other guest interrupts)
- Reduce exits required to inject a virtual interrupt
  - Use ELI to asynchronously inject virtual interrupts from a different core
- Improve performance of SMP workloads with high Inter-processor interrupt (IPI) rate
  - Send and IPI directly to a vcpu running on a different core without forcing an exit

Conclusions

- High virtualization performance requires the CPU to spend most of the time running the guest (useful work) and not the host (handling exits=overhead)
- x86 virtualization requires host involvement (exits!) to handle interrupts (critical path for I/O workloads)
- ELI lets the guest handle interrupts directly (no exits!) and securely, making it possible for untrusted and unmodified guests reach near bare-metal performance

Questions ?

Backup

Injection Mode and Nested Interrupts

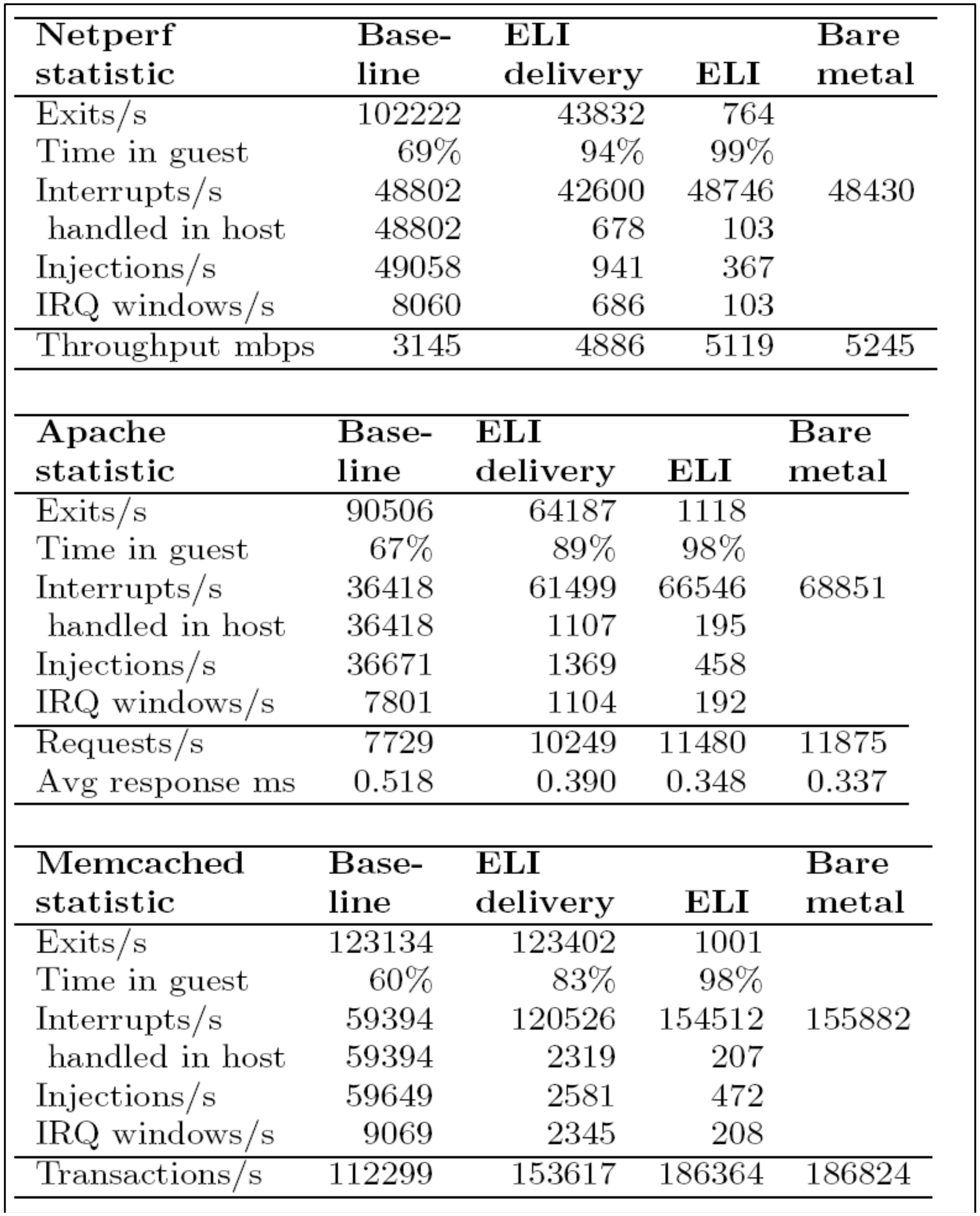
- ELI has two mode of operations:
  - Direct Mode: physical interrupts are delivered trough the shadow IDT and do not force an exit (only if NP#). MSR Bitmap is configured to avoid exits on EOI
  - Injection Mode: physical interrupts and EOI force an exit. Virtual interrupts are delivered through the Guest IDT

- The guest runs most of the time in Direct Mode

- The hypervisor fall-back to Inject Mode when it needs to inject a “virtual” interrupt
  - After the virtual EOI exit, the hypervisor switches back to Direct Mode

- A higher vector can interrupt a lower vector (Nested Interrupts)
  - Before Injecting a virtual interrupt ELI checks the CPU interrupts in service register (ISR)
  - If the virtual interrupt has a lower vector than a physical interrupt being handled, the hypervisor delays the injection.

Breakdown

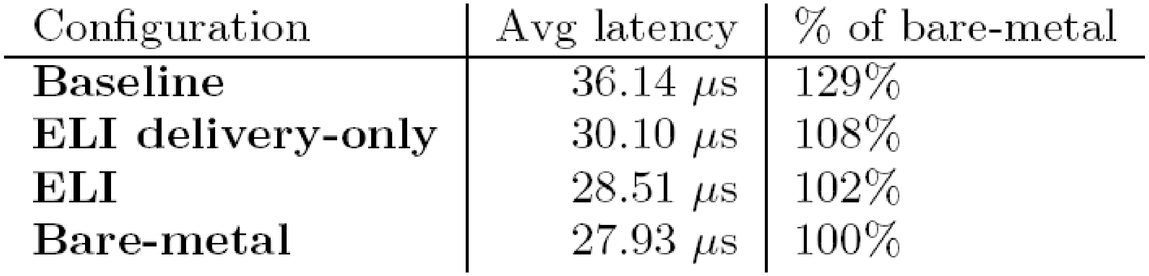


Netperf statistic	Base- line	ELI delivery	ELI	Bare metal
Exits/s	102222	43832	764	
Time in guest	69%	94%	99%	
Interrupts/s	48802	42600	48746	48430
handled in host	48802	678	103	
Injections/s	49058	941	367	
IRQ windows/s	8060	686	103	
Throughput mbps	3145	4886	5119	5245

Apache statistic	Base- line	ELI delivery	ELI	Bare metal
Exits/s	90506	64187	1118	
Time in guest	67%	89%	98%	
Interrupts/s	36418	61499	66546	68851
handled in host	36418	1107	195	
Injections/s	36671	1369	458	
IRQ windows/s	7801	1104	192	
Requests/s	7729	10249	11480	11875
Avg response ms	0.518	0.390	0.348	0.337

Memcached statistic	Base- line	ELI delivery	ELI	Bare metal
Exits/s	123134	123402	1001	
Time in guest	60%	83%	98%	
Interrupts/s	59394	120526	154512	155882
handled in host	59394	2319	207	
Injections/s	59649	2581	472	
IRQ windows/s	9069	2345	208	
Transactions/s	112299	153617	186364	186824

Latency



Configuration	Avg latency	% of bare-metal
Baseline	36.14 μs	129%
ELI delivery-only	30.10 μs	108%
ELI	28.51 μs	102%
Bare-metal	27.93 μs	100%

Latency – Netperf UDP RR