

# Understanding PCIe performance for end host networking

*Rolf Neugebauer*, Gianni Antichi, José Fernando Zazo,  
Yury Audzevich, Sergio López-Buedo, Andrew W. Moore



# The idea of end hosts participating in the implementation of network functionality has been extensively explored in enterprise and datacenter networks

**Enabling End-host Network Functions**  
 Vasiliki Angelaki, Christos Cikalidis, Matthew P. Grosvenor, George Koromilas, and Greg O'Shea  
 Microsoft Research, UK  
 SDN, Network Management, Network Functions

## Fabric: A Retrospective on Evolving SDN

Martin Casado, Teemu Koponen, Scott Shenker  
 Microsoft, Intel, ICSI/UC Berkeley

**Abstract**  
 MPLS was an attempt to retrofit network hardware with layers for the flexibility of network control. Software-Defined Networking (SDN) was designed to enable further progress along both of these dimensions. While a significant step forward in some respects, it was a step backward in others. In this paper we discuss SDN shortcomings and propose how they can be overcome by the insight underlying MPLS. We believe this hybrid approach enables an era of simple hardware and flexible control.

**Categories and Subject Descriptors**  
 C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks—Internet; C.2.1 [Computer-Communication Networks]: Network Architecture and Design

**Extending Networking into the Virtualization Layer**  
 Teemu Koponen, Scott Shenker, Justin Pettit, Martin Casado, Ben Pfaff, Keith Amundson

**ABSTRACT**  
 The move to virtualization has created a new network access layer residing on hosts that connects the various VMs. Virtualized environments impose requirements on networking for which traditional models are not well suited. They also provide advantages to the networking layer (such as software flexibility and well-defined end host events) that are not present in physical networks. To date, this new virtualization network layer has been largely built around standard Ethernet networking, but also technology with other advantages. We present Open vSwitch, a network switch specifically built for virtual environments. Open vSwitch differs from traditional approaches in that it exports an external interface for state-based control of configuration state and forwarding behavior. We describe how Open vSwitch can be used in a variety of scenarios such as inclusion in

**ABSTRACT**  
 Enterprise network architecture and management has followed the Internet's design principles despite different requirements that business applications are subjected to a single set of business applications. We address a new approach where each application is treated as a separate entity within the network's complex, but abstract, architecture. To enable this approach, we propose a new network layer that is composed of the following components:

**Network Exception Handlers: Host-network Control in Enterprise Networks**  
 Thomas Karagiannis, Richard Mortier and Antony Rowstron  
 (thomkar, antrow)@microsoft.com, mort@wyplodia.com  
 Microsoft Research, Cambridge, UK

## SideCar: Building Programmable Datacenter Networks without Programmable Switches

Alan Shieh<sup>1</sup>, Srikanth Kandula<sup>1</sup>, Emlin Gun Sirer<sup>2</sup>  
<sup>1</sup> Microsoft Research and <sup>2</sup> Cornell University

**Abstract**— This paper examines an extreme point in the design space of programmable switches and network policy enforcement. Rather than relying on extensive changes to switches to provide more programmability, SideCar distributes custom processing code between servers running on every end host and general purpose server processors, such as server blades, connected to each switch via commonly available redirection mechanisms. This provides applications with pervasive network instrumentation and programmability on the forwarding plane. While not a perfect replacement for programmable switches, this has several growing problems while requiring little or no change to existing switches. In particular, in the context of public cloud data centers with

general purpose server processors, but are otherwise minimally modified, i.e., no internal changes to the software or hardware of the switch. With these constraints, SideCar enables applications to install custom packet processing rules that execute within the network; these rules consist of a packet classifier, combined with associated code that processes every packet matching that classifier.

Our key insight in realizing this programming model entails pushing packet classification to the edge and offloading custom processing to commodity servers. By having end hosts intercept packets as needed, special processing and having switches redirect designated packets, the hardware requirements for switches are substantially reduced: each switch need only process a small set of packet classifiers rather than a large set of complex packet formats. By lim-

# More recently, programmable NICs and FPGAs enable offload and NIC customisation

**SENIC: Scalable NIC for End-Host Rate Limiting**  
 Sivasankar Radhakrishnan\*, Yilong Gong\*, Vimal Kumar Jayakumar\*,  
 Abhal Kaulani†, George Porter\*, Anita Vahala\*,  
 \*University of California, San Diego  
 {sivasankar, yilong, vimal}@cs.ucsd.edu  
 †Google Inc.  
 abhalni@google.com

Property	Hardware	Software
Scales to many classes	X	X
Works at high link speeds	X	X
Low CPU overhead	X	X
Precise rate enforcement	X	X
Supports hypervisor bypass	X	X
Supports flow control	X	X

Table 1: Pros and cons of current hardware and software-based

## Enabling End-host Network Functions

Ritesh Ballani, Paolo Costa, Christos Gkantsidis, Matthew P. Grosvenor,  
 Thomas Karagiannis, Lazaros Kerkiras, and Greg O'Shea  
 Microsoft Research

### ABSTRACT

Many network functions executed in modern datacenters, e.g., load balancing, application-level QoS, and congestion control, exhibit three common properties at the data plane: they need to access and modify state, to perform computations, and to access application semantics — this is critical since many network functions are best expressed in terms of application-level messages. In this paper, we argue that these three properties are a natural

### Keywords

Software Defined Networking, SDN, Network Management, Data-plane programming, Network Functions

### 1 Introduction

Recent years have seen a lot of investment in functionality deployed across datacenter networks. Network functions range from management tasks like routing

### Abstract

The recent surge of network I/O performance has put even more pressure on memory and software I/O processing subsystems. We argue that the primary reason for high memory and processing overheads in the inefficient use of these resources by current commodity servers interface cards (NICs). We propose FlexNIC, a flexible network DMA interface that can be used by operating systems and applications to reduce packet processing overheads. FlexNIC of

## High Performance Packet Processing with FlexNIC

Annoie Kaufmann†, Simon Peter\*, Naveen Kr. Sharma†,  
 Thomas Anderson†, Arvind Krishnamurthy†,  
 †University of Washington {arvind, naveen}@cs.washington.edu  
 \*The University of Texas at Austin simondp@cs.utexas.edu

## HyperLoop: Group-Based NIC-Offloading to Accelerate Replicated Transactions in Multi-Tenant Storage Systems

Daehyeon Kim<sup>1\*</sup>, Amirreza Memarpour<sup>2\*</sup>, Anirudh Badam<sup>3</sup>,  
 Yibo Zhu<sup>3</sup>, Hongqi Harry Liu<sup>1\*</sup>, Jitu Padhye<sup>3</sup>, Shachar Rindell<sup>3</sup>,  
 Steven Swanson<sup>2</sup>, Vyas Sekar<sup>1</sup>, Srinivasan Seshan<sup>1</sup>  
<sup>1</sup>Carnegie Mellon University, <sup>2</sup>UC San Diego, <sup>3</sup>Microsoft

### ABSTRACT

Storage systems in data centers are an important component of large-scale online services. They typically perform replicated transactional operations for high data availability and

### CCS CONCEPTS

• Networks → Data center networks; • Information systems → Remote replication; • Computer systems organization → Cloud computing.

## KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC

Bijie Li<sup>1\*</sup>, Zhenyuan Ruan<sup>1\*</sup>, Wencong Xiao<sup>1\*</sup>, Yuanwei Lu<sup>1†</sup>,  
 Yongqiang Xiong<sup>1</sup>, Andrew Putnam<sup>1</sup>, Enhong Chen<sup>2</sup>, Lintao Zhang<sup>2</sup>  
<sup>1</sup>Microsoft Research, <sup>2</sup>USTC, <sup>3</sup>UCLA, <sup>4</sup>Beihang University

### ABSTRACT

Use of in-memory key-value store (KVS) continues to gain importance as modern KVS goes beyond the traditional main memory and becomes a first-class

### CCS CONCEPTS

• Information systems → Key-value stores; • Hardware

- Isolation
- QoS
- Load balancing
- Application specific processing
- ....

# Not “just” in academia, but in production!

## **Azure Accelerated Networking: SmartNICs in the Public Cloud**

Daniel Firestone Andrew Putnam Sambhrama Mundkur Derek Chiou Alireza Dabagh  
Mike Andrewartha Hari Angepat Vivek Bhanu Adrian Caulfield Eric Chung  
Harish Kumar Chandrappa Somesh Chaturmohta Matt Humphrey Jack Lavier Norman Lam  
Fengfen Liu Kalin Ovtcharov Jitu Padhye Gautham Popuri Shachar Raindel Tejas Sapre  
Mark Shaw Gabriel Silva Madhan Sivakumar Nisheeth Srivastava Anshuman Verma Qasim Zuhair  
Deepak Bansal Doug Burger Kushagra Vaid David A. Maltz Albert Greenberg

Microsoft

### **Abstract**

Modern cloud architectures rely on each server running its own networking stack to implement policies such as tunneling for virtual networks, security, and load balancing. However, these networking stacks are becoming increasingly complex as features are added and as network speeds

all virtual networking features, such as private virtual networks with customer supplied address spaces, scalable L4 load balancers, security groups and access control lists (ACLs), virtual routing tables, bandwidth metering, QoS, and more. These features are the responsibility of the host platform, which typically means software running in the hypervisor.

# Implementing offloads is not easy

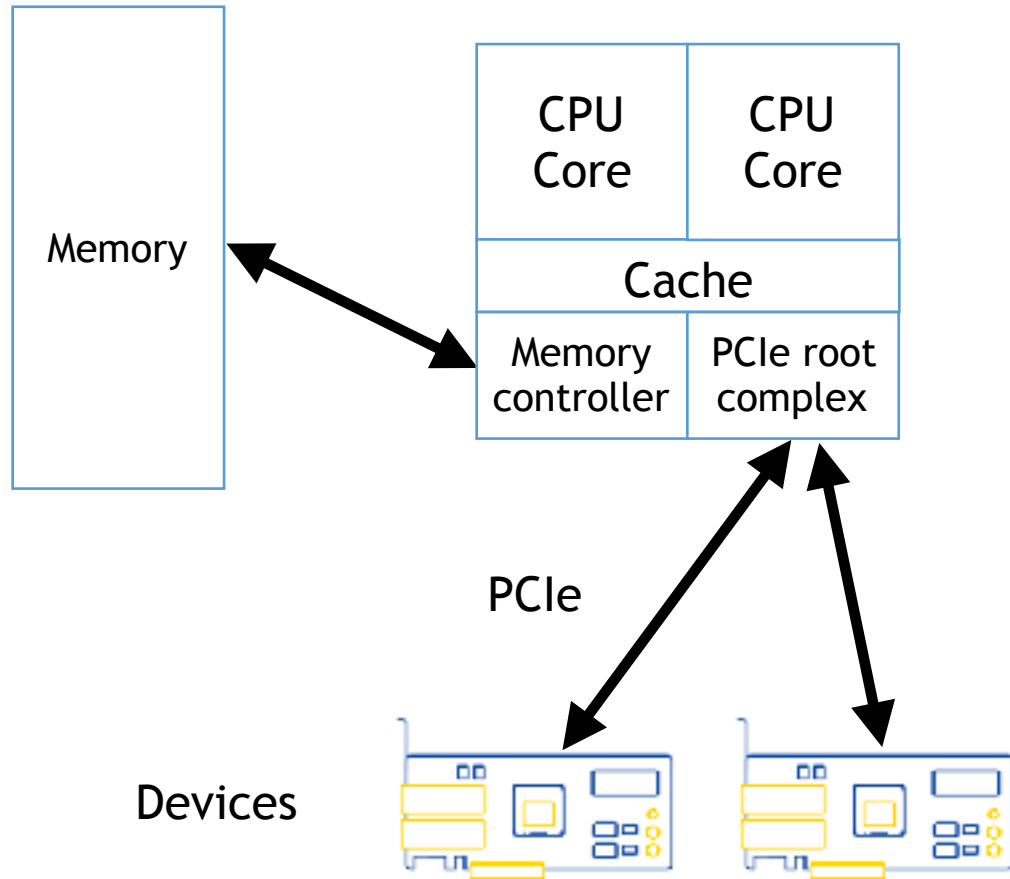
Many potential bottlenecks

# Implementing offloads is not easy

Many potential bottlenecks

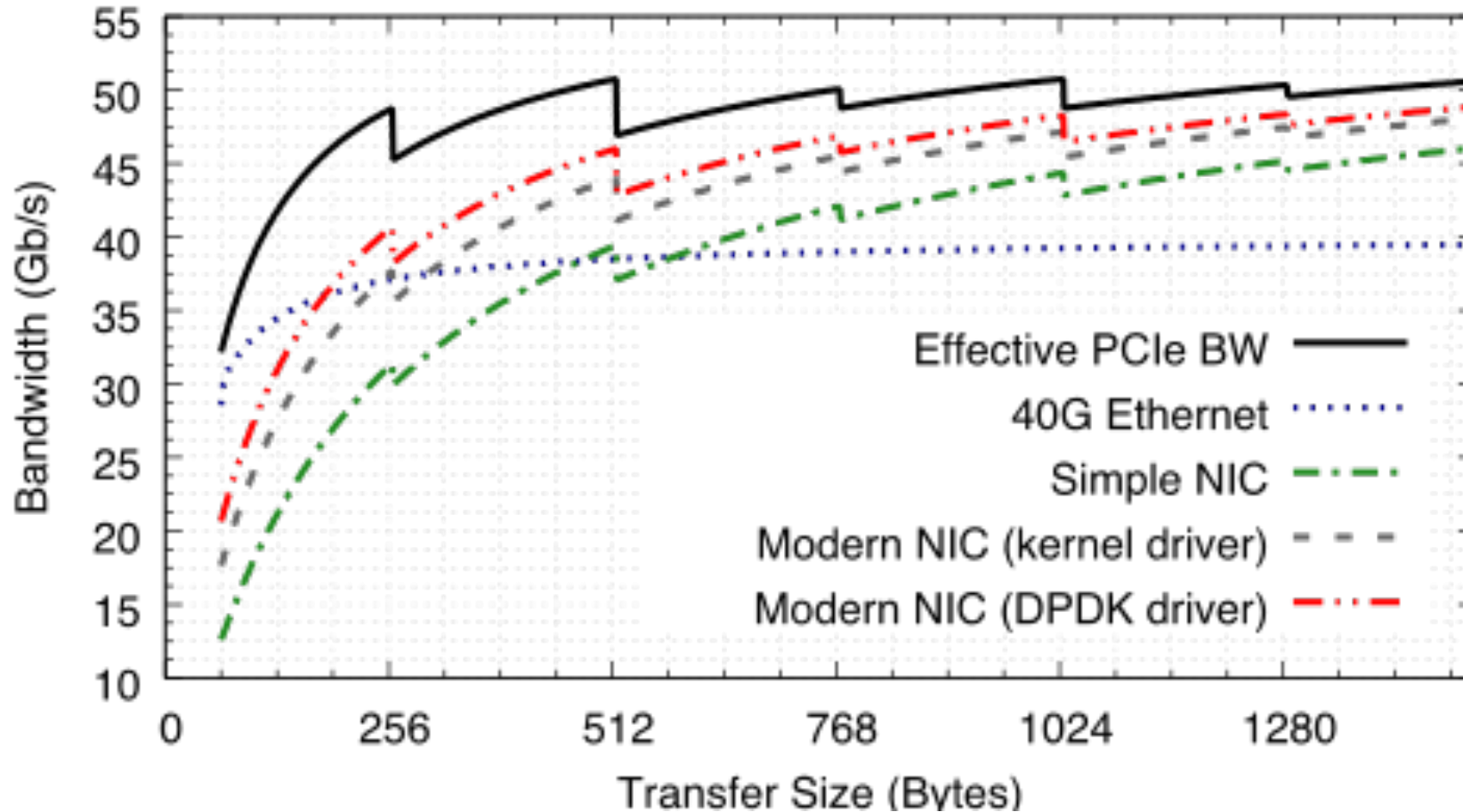
PCI Express (PCIe) and its implementation by the host  
is one of them!

# PCIe overview



- De facto standard to connect high performance IO devices to the rest of the system. Ex: NICs, NVMe, graphics, TPUs
- PCIe devices transfer data to/from host memory via **DMA** (direct memory access)
- **DMA engines** on each device translate requests like “Write these 1500 bytes to host address 0x1234” into multiple PCIe Memory Write (MWr) “packets”.
- PCIe is almost like a network protocol with packets (TLPs), headers, MTU (MPS), flow control, addressing and switching (and NAT ;)

# PCIe protocol overheads



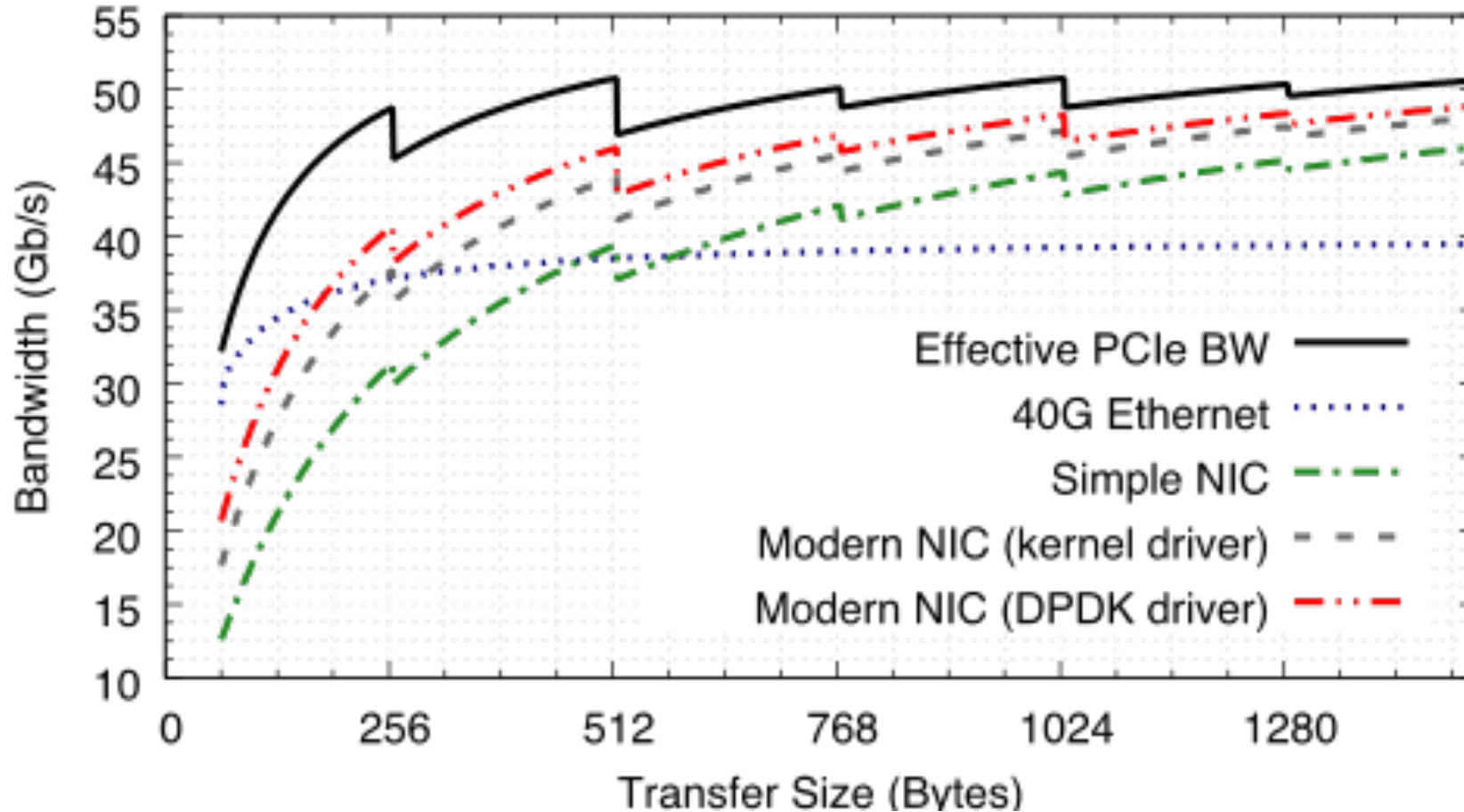
62.96 Gb/s at the physical layer

PCIe protocol

~ 32 - 50 Gb/s for data transfers



# PCIe protocol overheads



62.96 Gb/s at the physical layer

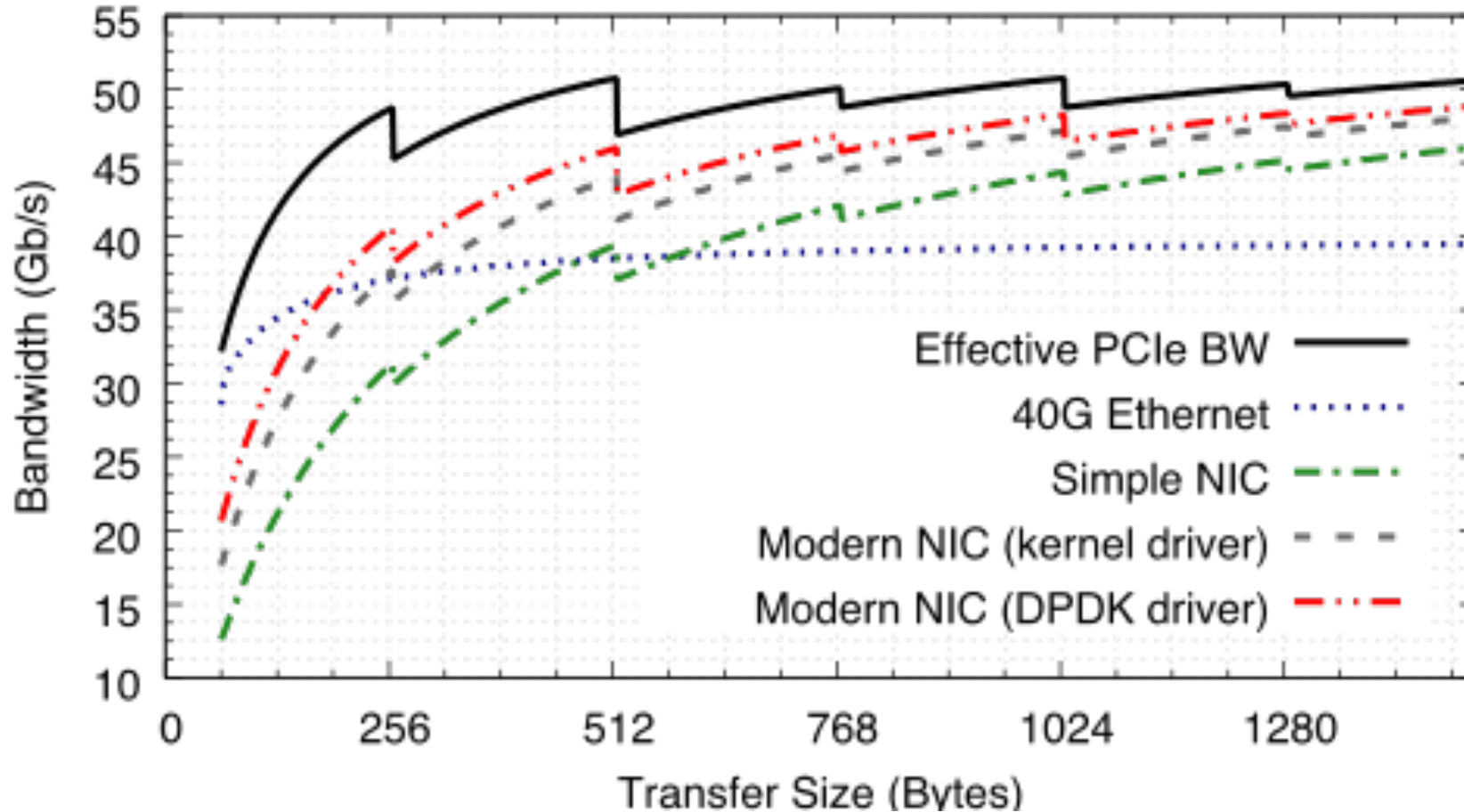
PCIe protocol

~ 32 - 50 Gb/s for data transfers

Queue pointer updates, descriptors, interrupts

~ 12 - 48 Gb/s

# PCIe protocol overheads



62.96 Gb/s at the physical layer

↓ PCIe protocol

~ 32 - 50 Gb/s for data transfers

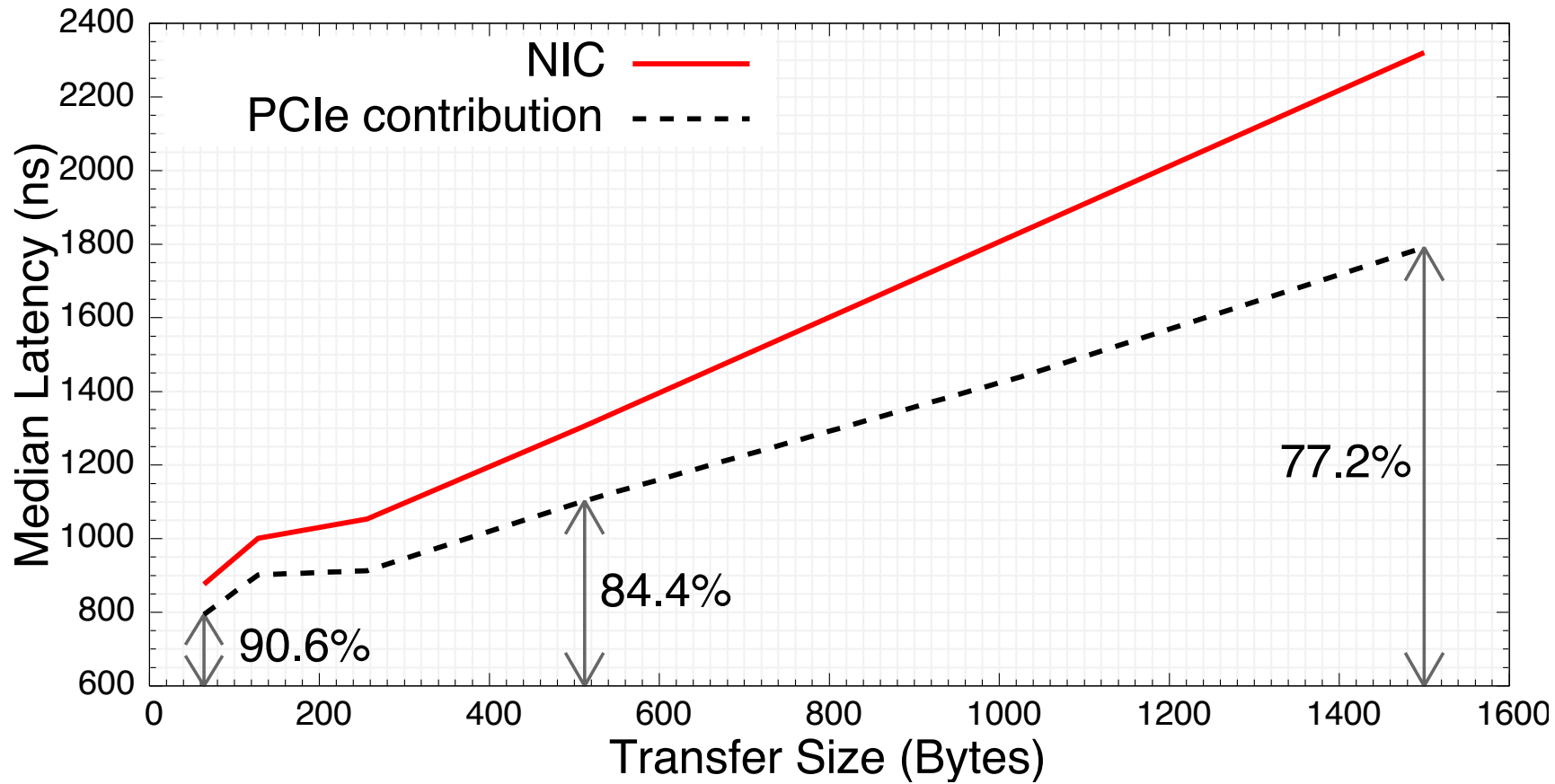
↓ Queue pointer updates, descriptors, interrupts

~ 12 - 48 Gb/s

**Complexity!**

Model: PCIe gen 3 x8 64 bit addressing

# PCIe latency

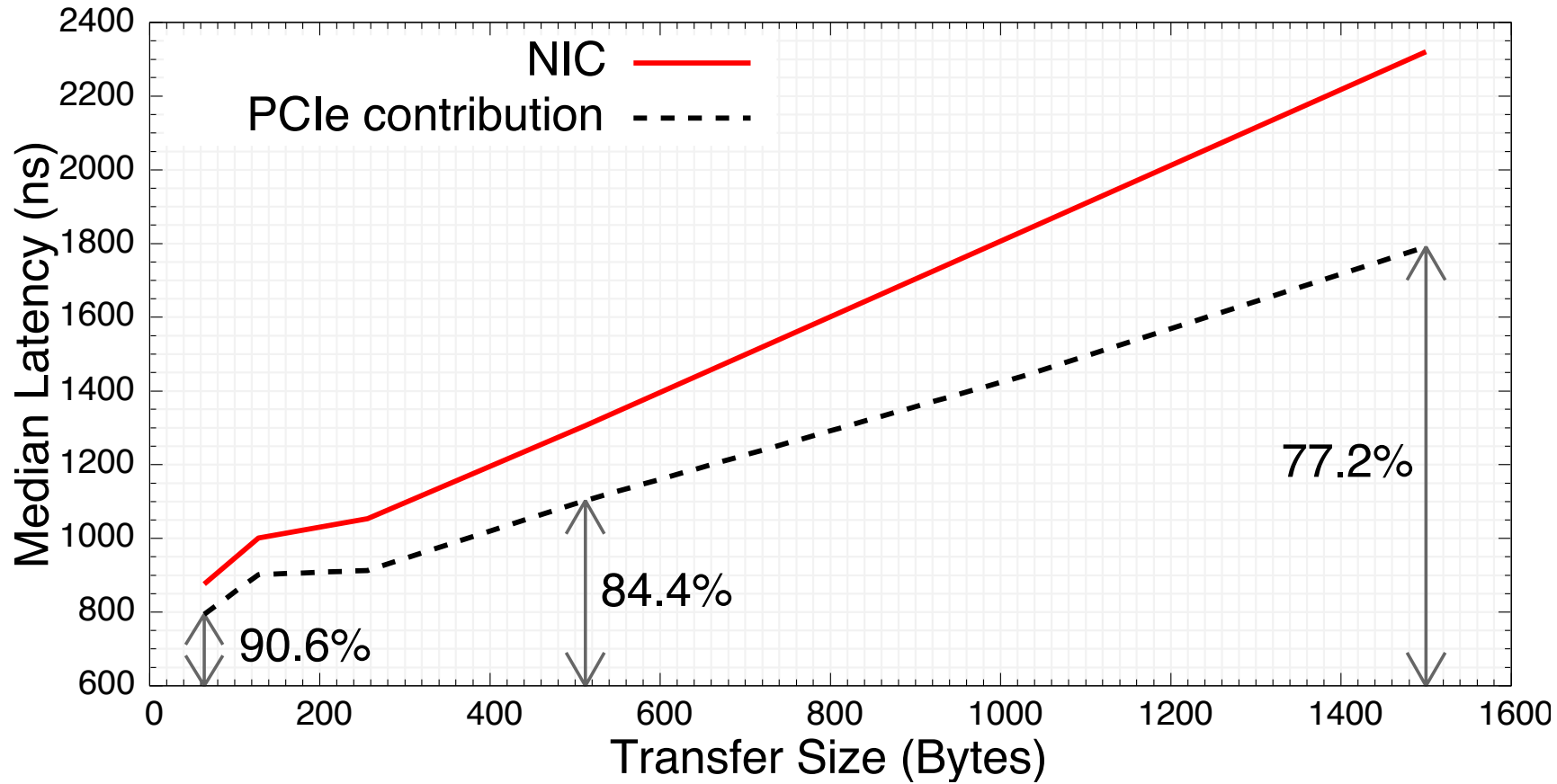


ExaNIC round trip times (loopback) with kernel bypass

PCIe contributes the majority of latency

Homa [SIGCOMM2018]:  
Desire single digit us latency for small messages

# PCIe latency imposes constraints

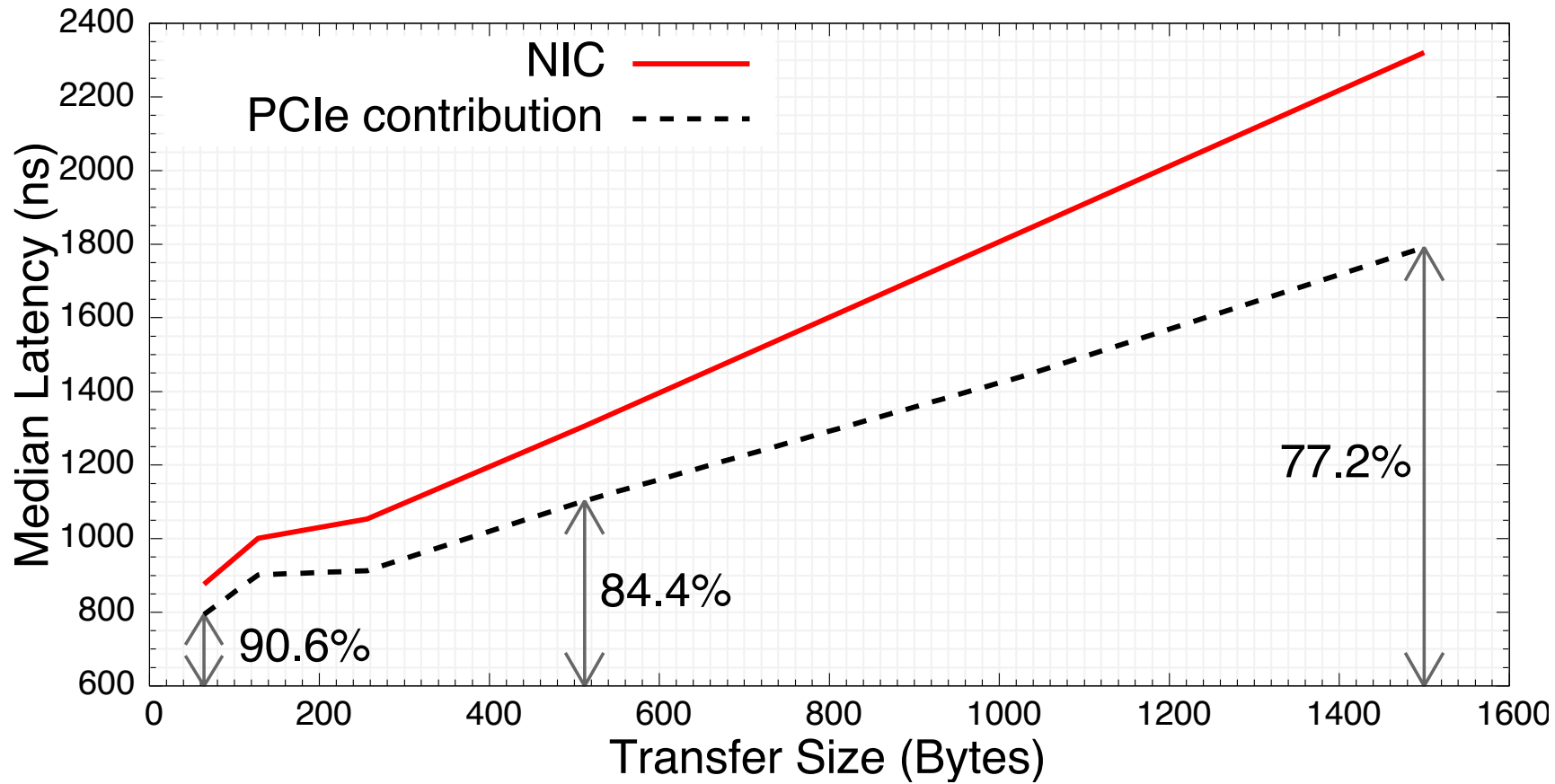


Ethernet line rate at 40Gb/s for 128B packets means a **new packet every 30ns.**

=

NIC has to handle at least 30 concurrent DMAs in each direction plus descriptor DMA

# PCIe latency imposes constraints



Ethernet line rate at 40Gbps for 128B packets means a **new packet every 30ns.**

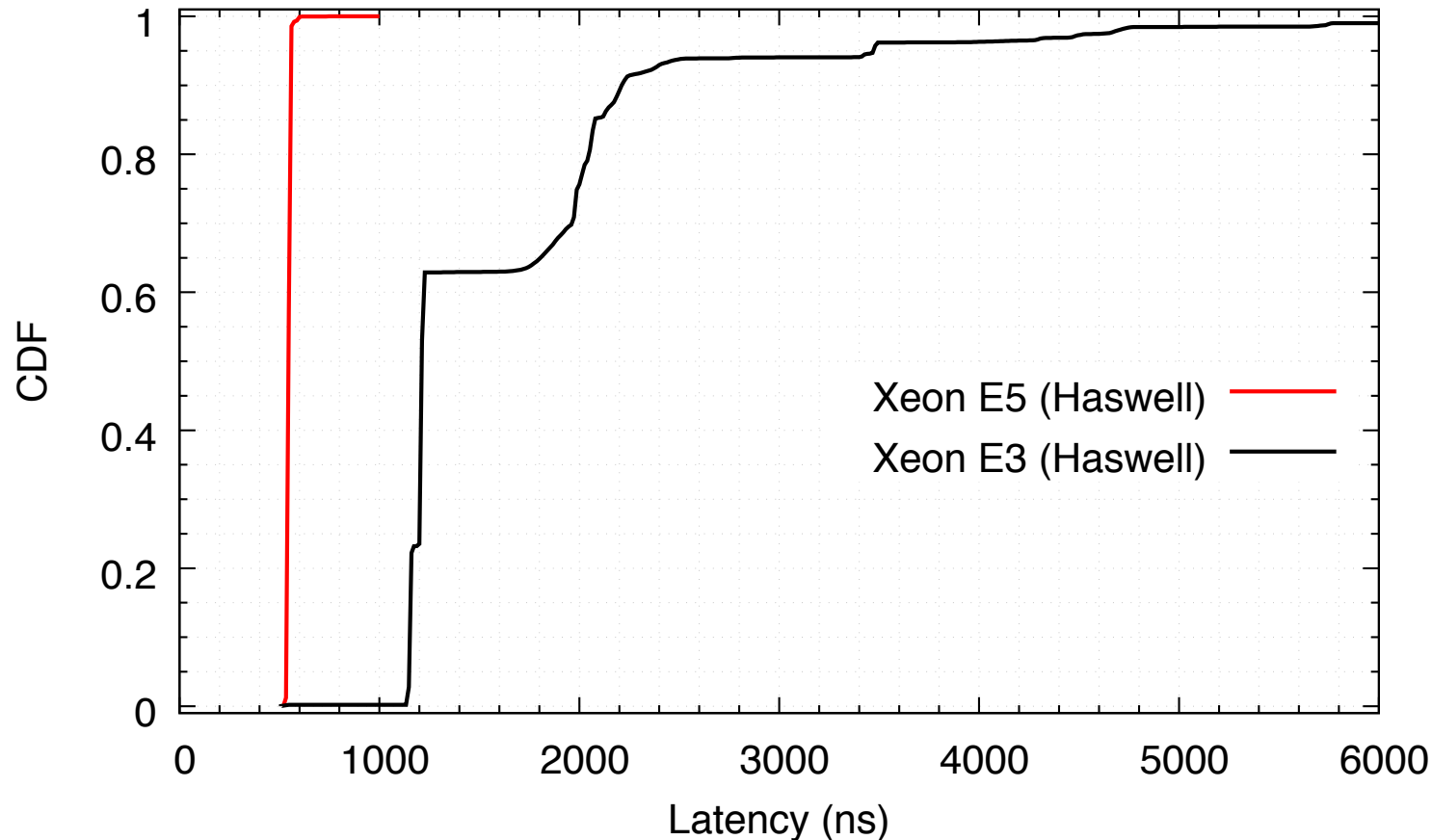
=

NIC has to handle at least 30 concurrent DMAs in each direction plus descriptor DMA

**Complexity!**

It get's worse...

# Distribution of 64B DMA Read latency



## Xeon E5

- 547ns median
- 573ns 99th percentile
- 1136ns max

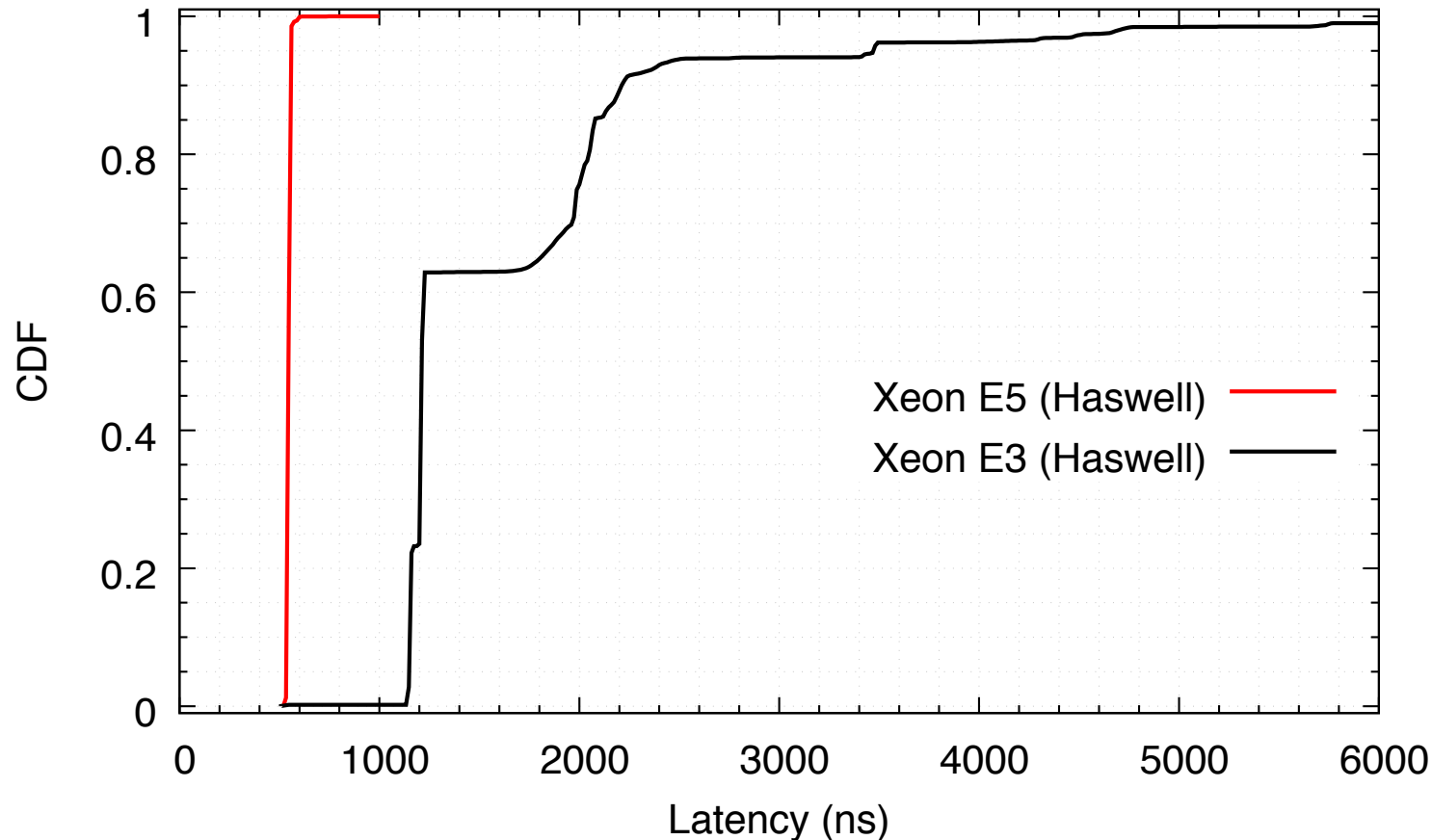
## Xeon E3

- 1213ns(!) median
- 5707ns(!) 99th percentile
- 5.8ms(!!!) max

Netronome NFP-6000, Intel Xeon E5-2637v3 @ 3.5GHz (Haswell)

Netronome NFP-6000, Intel Xeon E3-1226v3 @ 3.3GHz (Haswell)

# Distribution of 64B DMA Read latency



## Xeon E5

- 547ns median
- 573ns 99th percentile
- 1136ns max

## Xeon E3

- 1213ns(!) median
- 5707ns(!) 99th percentile
- 5.8ms(!!!) max

**Your offload implementation has to handle this!**

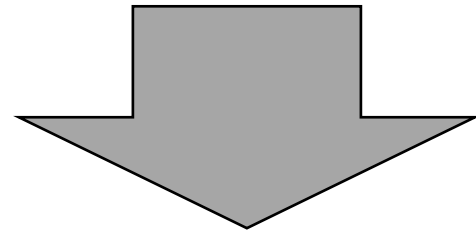
Netronome NFP-6000, Intel Xeon E5-2637v3 @ 3.5GHz (Haswell)

Netronome NFP-6000, Intel Xeon E3-1226v3 @ 3.3GHz (Haswell)



# PCIe host implementation is evolving

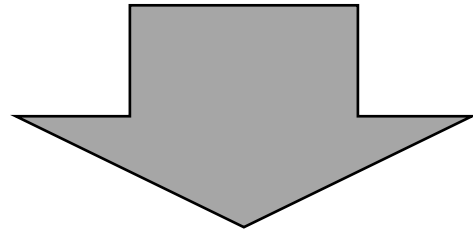
- Tighter integration of PCIe and CPU caches (e.g. Intel's **DDIO**)
- PCIe device is local to some memory (**NUMA**)
- **IOMMU** interposed between PCIe device and host memory



PCIe transactions are dependent on temporal state on the host and the location in host memory

# PCIe host implementation is evolving

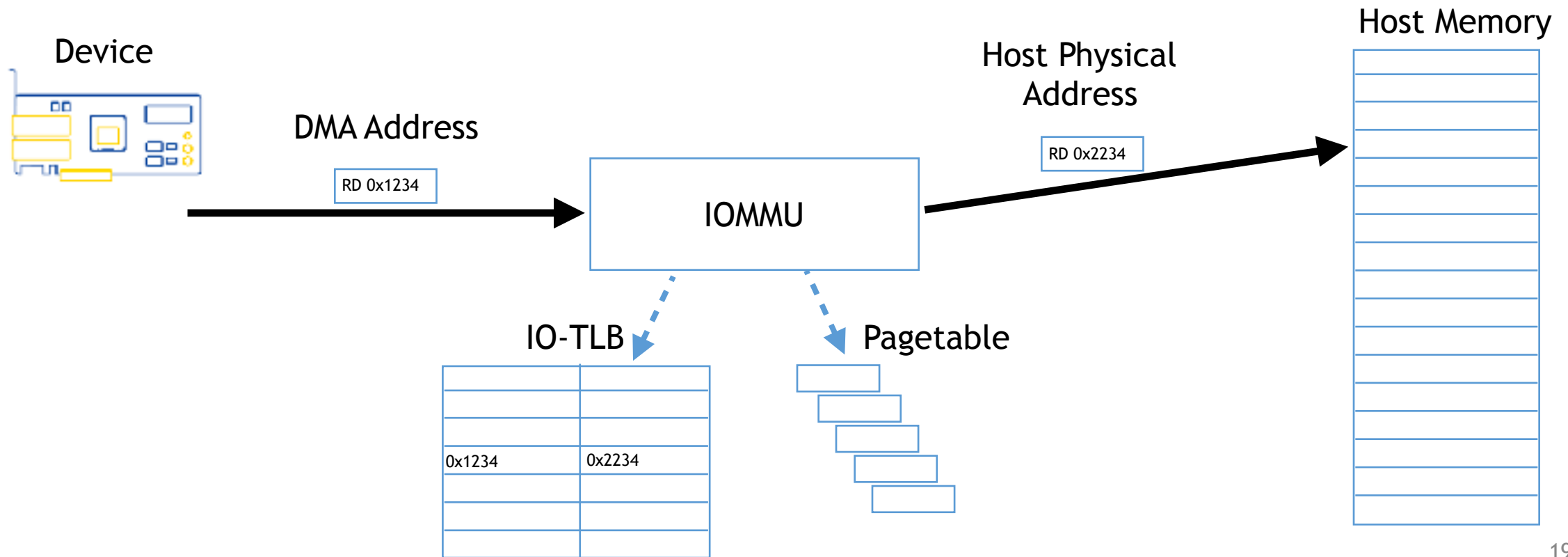
- Tighter integration of PCIe and caches (e.g. Intel's **DDIO**)
- PCIe is local to some memory (**NUMA**)
- **IOMMU** interposed between PCIe device and host memory



PCIe transactions are dependent on temporal state on the host and the location in host memory

# PCIe data-path with IOMMU (simplified)

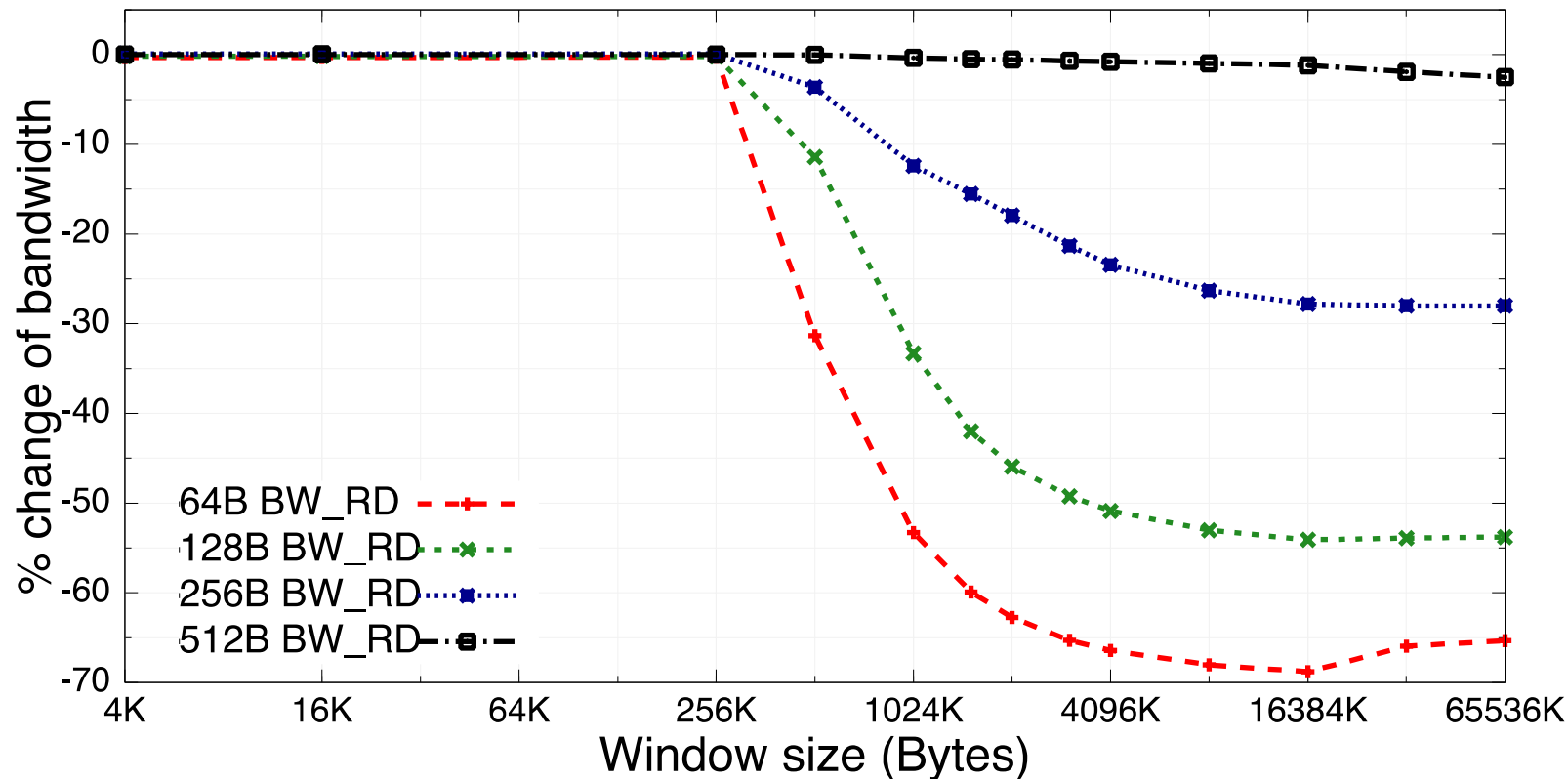
- IOMMUs translate addresses in PCIe transactions to host addresses
- Use a Translation Lookaside Buffer (TLB) as cache
- On TLB miss, perform a costly pageable walk, replace TLB entry



# Measuring the impact of the IOMMU

- DMA reads of fixed size
- From random addresses on the host
- Systematically change the address range (window) we access
- Measure achieved bandwidth (or latency)
- Compare with non-IOMMU case

# IOMMU results



- Different transfer sizes
- Throughput drops dramatically once region exceeds 256K.
- TLB thrashing
- TLB has 64 entries (256KB/4096B)  
Not published by Intel!
- Effect more dramatic for smaller transfer sizes

# Understanding PCIe performance is important

- A plethora of tools exist to analyse and understand OS and application performance
  - ... but very little data available on PCIe contributions
- Important when implementing offloads to programmable NICs
  - ... but also applicable to other high performance IO devices such as ML accelerators, modern storage adapters, etc

# Introducing **pcie-bench**

- A **model** of PCIe to quickly analyse **protocol** overheads
- A suite of **benchmark tools** in the spirit of lmbench/hbench
- Records latency of individual transactions and bandwidth of batches
- Allows to systematically change
  - Type of PCIe transaction (PCIe read/write)
  - Transfer size of PCIe transaction
  - Offsets for host memory address (for unaligned DMA)
  - Address range and NUMA location of memory to access
  - Access pattern (seq/rand)
  - State of host caches
- ▶ Provides detailed insights into PCIe host and device implementations

# Two independent implementations

- Netronome NFP-4000 and NFP-6000
  - Firmware written in Micro-C (~1500 loc)
  - Timer resolution 19.2ns
  - Kernel driver (~400 loc) and control program (~1600 loc)
- NetFPGA and Xilinx VC709 evaluation board
  - Logic written in Verilog (~1200 loc)
  - Timer resolution 4ns
  - Kernel driver (~800 loc) and control program (~600 loc)

[implementations on other devices possible]



# Conclusions

- The **PCIe protocol** adds significant overhead esp for small transactions
- **PCIe implementations** have a significant impact on IO performance:
  - Contributes significantly to the latency (70-90% on ExaNIC)
  - Big difference between two the implementations we measured (what about AMD, arm64, power?)
  - Performance is dependent on temporal host state (TLB, caches)
  - Dependent on other devices?
- Introduced **pcie-bench** to
  - understand PCIe performance in detail
  - aid development of custom NIC offload and other IO accelerators
- Presented the first detailed study of PCIe performance in modern servers

# Thank you!

Source code and all the data is available at:

<https://www.pcie-bench.org>

<https://github.com/pcie-bench>