PCI Express as a Killer of Software-based Real-Time Ethernet

Rostislav Lisový, Michal Sojka, Zdeněk Hanzálek

Czech Technical University in Prague, Faculty of Electrical Engineering
Technická 2, 121 35 Prague 6, Czech Republic
Email: {lisovros,sojkam1,hanzalek}@fel.cvut.cz

8th July 2013
Realtime Ethernet

Why to choose Ethernet?
- Mature technology
- High bandwidth
- Low cost

How to make it Realtime (i.e. deterministic)?
- Token passing
- Bandwidth limitation
- TDMA – Time-triggered

Realtime properties ensured either by Hardware or Software
Software implementations of a Time-Triggereed Ethernet (TTE) protocols

Reasons to implement and use a software stack

- Cost reduction
- Rapid prototyping

Already available software TTE protocol stacks

- RTnet
- TTTech TTE Protocol Layer
- Sysgo AFDX software node
Software implementations of a Time-Triggered Ethernet (TTE) protocols

Reasons to implement and use a software stack

- Cost reduction
- Rapid prototyping

Already available software TTE protocol stacks

- RTnet
- TTTTech TTE Protocol Layer
- Sysgo AFDX software node

Drawbacks

- Uncertainties in timing (latencies, jitter) require longer inter-frame gaps.
- Longer inter-frame gaps ⇒ lower efficiency.
Latencies in software TTE implementations
... and how to get rid of them

▶ Protocol implementation
  ▶ Implementation quality
Latencies in software TTE implementations
... and how to get rid of them

- Protocol implementation
  - Implementation quality
- Operating system (cca. 30 $\mu$s)
  - OS scheduler latencies
  - Unrelated software components (e.g. drivers) may disable interrupts
  - We use NOVA to avoid OS latencies and provide isolation
Latencies in software TTE implementations
... and how to get rid of them

- Protocol implementation
  - Implementation quality
- Operating system (cca. 30 $\mu$s)
  - OS scheduler latencies
  - Unrelated software components (e.g. drivers) may disable interrupts
    - *We use NOVA to avoid OS latencies and provide isolation*
- Hardware latencies
  - *Often neglected [1]*
  - How big are they?

Outline

Testbed setup
  - Software architecture
  - Used hardware
  - Modern PC architecture

Experiments
  - Measurement procedure
  - Conducted experiments

Conclusion
Software architecture

NOVA microhypervisor
- Originally developed in TU Dresden, Germany; Now Intel Labs
- Microkernel, x86(-64) only
- Very small code base (9 kLoC)
- Virtualisation support

Time-triggered protocol implementation
- One CPU core fully dedicated to the application
- Code and data fit into L2 cache of the CPU
- During experiments, VMs running on another CPUs
Used hardware

Off-the-shelf x86 quad-core personal computer (Ivy Bridge CPU)
  ▶ Low cost
  ▶ High availability

Intel 82576 Gigabit Ethernet controller (NIC)
  ▶ Connected to PCI Express
  ▶ IEEE 1588 (Precision Time Protocol) support
    ▶ Clock in NIC (16 ns resolution)
    ▶ Hardware timestamping of received/transmitted frames
Modern PC architecture

- PCIe – packet-based serial network
- PCIe for communication between CPU and peripherals
- Two different PCIe slot types – significant properties difference
NIC frame transmission

How to make the NIC to transmit the Ethernet frame

1. Store frame payload to RAM
2. Fill the Packet descriptor
3. Store the Packet descriptor to NIC
4. NIC fetches data from RAM
5. NIC starts transmitting to the Ethernet
Transmission latency

- $t_{TX}$ – time of transmission start captured by hardware (IEEE 1588 support) – high precision
- $t_{clk}$ – time of processing start measured by clock value readout – possible inaccuracy
Experiment 1: NIC clock readout latency

What was measured?

- $t_{clk2} - t_{clk1}$
**Experiment 1: NIC clock readout latency**

**Measured variants**

- **Different PCIe slots**
  - Core 0
  - Core 1
  - Core 2
  - Core 3

- **Different system loads**
  - VM 1
  - VM 2
  - VM 3
  - VM 4

---

**Diagram**

- CPU
- Root Complex
- RAM DDR3
- PCIe
- PCIe x16
- DMI / PCIe
- PCH (Chipset)
- HPET
- SATA
- USB
- LPC
- PCIe Slots
- PCIe Switch
- PCIe to PCI Bridge
- PCI Slots
- PCI Express Adapter Slot (GFX)
- PCIe x16
- Serial / Parallel port; Keyboard, Mouse PS/2

---

**Key Components**

- **NOVA microhypervisor**
- **VMM**
- **RTEth**
- **NIC driver**

**Booting Modes**

- **Kernel mode**
- **User mode**

**CPU Cores**

- CPU 0
- CPU 1
- CPU 2
- CPU 3

**NIC Configurations**

- VM 4
- VMM
- VM 1
- VMM
- VM 2
- VMM
- VM 3
- VMM
- RTEth + NIC driver

**System Blocks**

- **NOVA microhypervisor**
  - Hardware (HW)
  - System Load (Kernel mode)
  - User Mode
Experiment 1: NIC clock readout latency

Results

- No intentional load
  - **GFX slot**
    - Avg. latency: 1.38 $\mu$s
    - Jitter: 5.31 $\mu$s
  - **IO slot**
    - Avg. latency: 3.11 $\mu$s
    - Jitter: 1.87 $\mu$s

- CPU load
Experiment 2: NIC transmission latency

What was measured?

- $t_{TX} - t_{clk2}$
Experiment 2: NIC transmission latency

Results, GFX slot

GFX PCIe slot, different loads

- **No load** latencies range from 5.1 to 11 µs (jitter cca. 6 µs)
- **Worst-case** latencies range from 5 to 14 µs (jitter cca. 9 µs)
Experiment 2: NIC transmission latency

Results, IO slot

IO PCIe slot, different loads

- No load latencies range from 8.5 to 13.5 $\mu$s (jitter cca. 5 $\mu$s)
- Worst-case latencies range from 8.5 to 19.5 $\mu$s (jitter cca. 11 $\mu$s)
Conclusion

- Identified an often **neglected source of jitter** for software-based implementations of the time-triggered Ethernet protocols
- NIC transmission latency jitter up to 10 $\mu s$
- Hardware with the IEEE 1588 support is capable of synchronising time with sub-microsecond precision, however the achieved **frame transmission precision is much worse**
- Possible mitigation by using COTS NIC which are capable of transmitting frames in specific time which is defined in Packet descriptor (Intel i210)
Thank you for your attention!