Commutation de paquets logicielle sur routeurs PC multi-coeurs

Mickaël Hoerdt, Dept. of computer engineering
Université catholique de Louvain la neuve
mickael.hoerdt@uclouvain.be

In collaboration with:

N. Egi, P. Papadimitriou, L. Mathy (Lancaster Uni.)
M. Handley, A. Greenhalgh (UCL) F. Huici (NEC Europe)
Introduction
Talk overview

- Background.
- Switching based on Software Tree.
- Software Forwarding Path overhead reduction.
Background

• Traditionally, high performance networking devices:
  - Built as custom hardware
  - Made of various specialized “CPUs”
  - Specialized multi-cores

• Software routers
  - Familiar programming environment
  - Easily extensible
  - Cheap
  - Generic packet processing capability
  - Run on (slow ?) commodity hardware ?
But recent advances in commodity HW architectures

- Multi-core (With very high clock frequency)
- Buses (usual bottlenecks) are disappearing
- Memory closer to the CPU: fast and small
- Memory Further from the CPU: slow and big
- Huge memory latency difference between L1 and Main memory (up to a factor 100)
Background

• Clicks Elements Scheduling

- Elements are C++ classes.
- Only PollDevice and ToDevice are scheduled.
- The others elements are scheduled indirectly via virtual C++ function calls: push() and pull().
- An element instance can only be executed by one core at a time.
• Clicks Elements Scheduling

• Elements are C++ classes.
• Only PollDevice and ToDevice are scheduled
• The others elements are scheduled indirectly via virtual C++ function calls : push() and pull().
• An element instance can only be executed by one core at a time.
To keep the data plane performances as high as possible:

- Distribute the computation among the CPU cores to maximize the number of CPU cycles.

- Keep the packets as deep as possible inside the cache hierarchy to minimize memory latency.

- How do we process packets and always keep the packets as deep as possible in the cache hierarchy and maximize the number of available CPU cycles?

- Let's have a look at the switching case.
Switching based on a Software Tree
Switching based on a Software Tree

The diagram illustrates the switching process based on a software tree. The flow involves PollDevice, Lookup, and ToDevice components connected through a series of layers labeled L1 and L2. The diagram highlights the core components for CORE 1 and CORE 2, with NICs connected at various points. Spinlocks are indicated at the core interfaces, suggesting synchronization points in the software tree.
System under test

- **Dell PowerEdge 2950 [SMP architecture]**
  - 2 Intel Xeon X5355 CPUs (quad-core @2.66GHz)
    - 32K L1d, 32K L1i, 4M L2 shared d&i cache (shared by 2 cores)
  - 8*1GB system memory
    - PC-5300
  - 12 Gigabit Ethernet ports
    - On 3 NICs
    - Each NIC had 4 PCIe
    - Intel 82571EB
  - Software
    - Click in Linux kernel
    - Xen VMM/Openvz
    - Oprofile, etc.
Switching based on a Software Tree

• Baseline

- Problem: lock contention overhead on the outgoing interface
- Increasing the number of cores increases the contention
Software Forwarding Path Overhead reduction

- Lets try to reduce the lock contention
  - ToDevices elements are contending on the lock
  - Grabbing the lock is costly (it's in main memory)
    - Lets avoid to schedule ToDevices when there are no packets to process in the upstream queue.
    - Consequence: only the Click elements that are processing packets will touch the main memory.
Software Forwarding Path Overhead reduction

Scénario A

Scénario B
Software Forwarding Path Overhead reduction

- Inactive forwarding paths are not scheduled.
- The locks are staying in the CPU cache.
= Increased performances.
Software Forwarding Path Overhead reduction

- If L2 is not shared and queue test: performances become worse! (-13 %)
- If L2 is not shared and queue test: performances become worse! (-13 %)
- Minus 15% of difference between the ideal result (A1) and shared L2 with queue testing
**Scénario A**

- If L2 is not shared and queue test: performances become worse! (-13%)

**Scénario B**

- Minus 15% of difference between the ideal result (A1) and shared L2 with queue testing

**Scénario C**

- Between the worse and the best perf: -33% of difference.
Software Forwarding Path Overhead reduction

• Lets try to reduce the lock contention

  – It clearly comes from several ToDevices contending on the lock for the same outgoing interface.

• What about having only one ToDevice per outgoing interfaces then?
• One ToDevice per interface:
  – Trees are now going up to the software enqueueing.
Software Forwarding Path Overhead reduction

-30% of difference when a packet switch cache hierarchy
Software Forwarding Path Overhead reduction

Scenario A

Scenario B

Scenario C

Scenario A1

-27% of difference when adding lock contention without shared L2
- About 10% of difference between the ideal result (A1') and shared L2 with queue testing (C'')
Problem: If more than 2 outgoing interfaces packets can't avoid to switch cache hierarchies
• Solution: one ToDevice per interface per cache hierarchy:

Software Forwarding Path Overhead reduction
- Now only 300 kpps (10%) of difference between the ideal result (E') and One TD per output per cache hierarchy (D''''')
Conclusions & Future work

• Keeping the number of memory accesses low and processing packet by using all the cores at the same time is difficult.

• Switching packets using a tree is a convenient scheduling unit for packet processing elements.

• Mitigating the lock contention can be done by carefully associating processing elements to cache hierarchies

• Future work: Explore the general case of processing elements that share a critical section between cores.
• Exploit the latest NIC trends (HW multiqueueing...)

28
System under test

- **Nehalem architecture**
  - 2 Intel Xeon 5560 CPUs (quad-core @2.8GHz)
    - 32K L1d, 32K L1i,
    - 256 KB L2 d&i cache each
    - 8MB L3 shared d&i cache (shared by all cores)

- 10 Gigabit Ethernet dual ports
  - On 2 NICs
  - Hardware MQ
  - Intel 82598EB

- Software
  - Click in Linux kernel
Software Forwarding Paths Architectures

- Tree based forwarding: Click implementation

(a) Configuration logicielle standard pour un routeur IP

(b) Configuration basée sur les arbres de commutations pour un routeur IP
A Software router can:
- Forward 64 byte packets at 7Mp/s (million packets / second).
- Be sufficient for edge networks.
- Be faster than the cheap CISCOs.

Spare CPU cycles.
A Software router can:
- Forward 64 byte packets at 7Mp/s (million packets / second).
- Be sufficient for edge networks.
- Be faster than the cheap CISCOs.

Spare CPU cycles.

Why not run several (virtual) routers on a single box?

(commodity x86 system)
Why now? Enabling technologies

- Virtual machines on regular modern x86 PC hardware
  - XEN, VMware, etc.
  - Both Intel and AMD are adding virtualization extensions to their processors

- Inexpensive and powerful x86 PC hardware
  - Forwards several Gbp/s even with min. sized packets
  - Sufficient capacity for most small and medium size businesses

- Future hardware trend
  - Massively Multi-core x86 PC architecture
  - Advanced multi-queue NICs
Challenges of virtual routers?

- Isolating VRs from each others.
- Minimise virtualization overhead.
- Fair scheduling of resources.
- Mapping CPU cores to VRs in high performance way.
Why now? Enabling technologies

- **XEN**
  - Para-virtualization
  - VMs with performance close to native hardware
  - Excellent hardware support
  - Exploits Intel’s and AMD’s virtualization technologies

- **Click Modular Router**
  - Running in the kernel
  - Simple and small elements
  - Plenty of elements ready to use
  - Implement new elements is simple
  - High performance (outperforms even native Linux)

- **Mature open source Control Planes**
  - Xorp, Quagga