CSE 534
Software Routers

Vyas Sekar
Stony Brook University

Based in part on content borrowed from:
Katerina Argyraki, Sylvia Ratnasamy
Assigned Reading

• Click
  – Kohler et al
  – Widely used
  – Your next project 😊

• RouteBricks
  – Dobrescu et al
  – SOSP best paper in 2009!
Outline

• Motivation for software routers

• Click design

• RB design
Building routers

• Fast

• Programmable
  – custom statistics
  – filtering
  – packet transformation
  – ...
Why programmable routers

• New ISP services
  – intrusion detection, application acceleration

• Simpler network monitoring
  – measure link latency, track down traffic

• New protocols
  – IP traceback, Trajectory Sampling, ...

Enable flexible, extensible networks
Today: fast or programmable

- Fast “hardware” routers
  - throughput: Tbps
  - little programmability

- Programmable “software” routers
  - processing by general-purpose CPUs
  - throughput < 10Gbps

Can we get the best of both worlds?
Outline

• Motivation for software router

• Click design

• RB design
Click overview

• Modular architecture
  – Router = composition of modules
  – Router = data flow graph

• Basic unit of processing = Element

• Three key components
  – Ports
  – Configuration
  – Method interfaces
Simple Tee Element
Two types of “connections”

• Push
  – Source element has finished processing
  – Sends it downstream
  – E.g., FromDevice

• Pull
  – Destination is ready to process
  – Initiates packet transfer
  – E.g., ToDevice
“Flow” of processing
// Declare three elements ...
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;

// ...and connect them together
src -> ctr;
ctr -> sink;

// Alternate definition using syntactic sugar
FromDevice(eth0) -> Counter -> Discard;
class NullElement: public Element { public:
    NullElement();
    const char *class_name() const { return "Null"; }
    NullElement *clone() const { return new NullElement; }
    const char *processing() const { return AGNOSTIC; }
    void push(int port, Packet *p) { output(0).push(p); }
    Packet *pull(int port) { return input(0).pull(); }
};
Other modules

• Packet Classification

• Scheduling

• Queueing

• Routing

• Many others (user contributed)

• What you write 😊
Receive livelock in practice
Hard to prioritize other work over interrupts

![Graph showing forwarding performance of unmodified kernel.](image)

**Fig. 2.** Forwarding performance of unmodified kernel.

Source: Mogul & Ramakrishnan, ToCS 96
Idea: Polling/Hybrid

• Under heavy load, disable the network card’s interrupts

• Use polling instead
  – Ask if there is more work once you’ve done the first batch

• Click paper we read – does pure polling

• Linux NAPI -- hybrid
Idea: Polling/Hybrid

- Device in interrupt mode
  - Interrupt Reception
  - Driver disables interrupts
    - netif_rx_schedule()

- Device in polling mode
  - Kernel polls device

- Process_backlog()=0
  - (no more data in the buffer)
  - Driver re-enables interrupts

- Process_backlog()=1
  - (buffer not empty yet)

Source: download.intel.com/design/intarch/PAPERS/323704.pdf
Outline

• Motivation for software router

• Click design

• RB design
Contributions

• **single server**: design rules for parallelizing packet processing across multiple cores
  – many parallelizations have high synchronization overheads
  – a 10x speedup in single-server routing performance

• **cluster**: interconnect design compatible with commodity server technology
  – hardware router interconnects depend on custom technology
  – scalable to high-end edge router speeds; *e.g.*, 320Gbps
A single-server router

- Sockets with cores
- Memory controllers (integrated)
- Point-to-point links (e.g., QPI)
- Network Interface Cards (NICs)
- Ports
- N router links
Packet processing in a server

Per packet,
1. core polls input port
2. NIC writes packet to memory
3. core reads packet
4. core processes packet 
   \textit{(address lookup, checksum, etc.)}
5. core writes packet to port
Lesson#1: **multi-core alone isn’t enough**

`Older` (2008)

- Cores
- Shared front-side bus
- Memory controller in `chipset`

Current (2009)

- Cores
- I/O hub
- Mem

Hardware need: avoid shared-bus servers
Lesson#2: on cores and ports

Problem: locking

Hence, rule: one core per port
Lesson#2: on cores and ports

Problem: cache misses, inter-core communication

Hence, rule: one core per packet
Lesson#2: on cores and ports

• two rules:
  – one core per port
  – one core per packet
• problem: often, can’t simultaneously satisfy both
• solution: use multi-Q NICs

Example: when #cores > #ports

× one core per packet
× one core per port
Multi-Q NICs

- feature on modern NICs (for virtualization)
  - port associated with multiple queues on NIC
  - NIC demuxes (muxes) incoming (outgoing) traffic
  - demux based on hashing packet fields (e.g., source+destination address)
Multi-Q NICs

• feature on modern NICs (for virtualization)

• repurposed for routing
  – rule: one core per port queue
  – rule: one core per packet
Multi-Q NICs

- feature on modern NICs (for virtualization)
- repurposed for routing
  - rule: one core per port
  - rule: one core per packet
- if #queues per port == #cores, can always enforce both rules
Lesson#3: book-keeping

• solution: batch packet operations
• problem: excessive per packet book-keeping overhead
  – NIC transfers packets in batches of `k`

1. core polls input port
2. NIC writes packet to memory
3. core reads packet
4. core processes packet
5. core writes packet to out port
and packet descriptors
Recap: routing on a server

Design lessons:

1. parallel hardware
   - at cores and memory and NICs

2. careful queue-to-core allocation
   - one core per queue, per packet

3. reduced book-keeping per packet
   - modified NIC driver w/ batching
Revisiting an old idea:
NSFNET’s NSS router (early ’90s)

“The National Science Foundation Network”, Chinoy and Braun, 1992
A cluster-based router today

interconnect?

10Gbps
A naïve solution

$N^2$ internal links of capacity $R$

problem: commodity servers cannot accommodate $N \times R$ traffic
Interconnecting servers

Challenges

- any input can send up to $R$ bps to any output
  - but need a low-capacity interconnect ($\sim NR$)
  - *i.e.*, fewer (<N), lower-capacity (<R) links per server

- must cope with overload
Overload

- Need to drop 20Gbps; (fairly across input ports)

- Problem: requires global state

- Drop at input servers?

- Problem: output might receive up to $N \times R$ traffic

- Drop at output server?

- Problem: output might receive up to $N \times R$ traffic

- Need to drop 20Gbps; (fairly across input ports)
Interconnecting servers

Challenges

– any input can send up to R bps to any output
  • but need a lower-capacity interconnect
  • *i.e.*, fewer (<N), lower-capacity (<R) links per server

– must cope with overload
  • need distributed dropping without global scheduling
  • processing at servers should scale as R, not NxR
Interconnecting servers

Challenges

– any input can send up to $R$ bps to any output
– must cope with overload

With constraints (due to commodity servers and NICs)

– internal link rates $\leq R$
– per-node processing: $cR$ (small $c$)
– limited per-node fanout

Solution: Use Valiant Load Balancing (VLB)
Valiant Load Balancing (VLB)

- Valiant et al. [STOC’81], communication in multi-processors
- applied to data centers [Greenberg’09], all-optical routers [Kesslassy’03], traffic engineering [Zhang-Shen’04], etc.
- idea: random load-balancing across a low-capacity interconnect
Packets arriving at external port:
- $N^2$ internal links of capacity $R/N$
- each server receives up to $R$ bps

Packets forwarded in two phases:

---

**phase 1**

- $N^2$ internal links of capacity $R/N$
- each server receives up to $R$ bps traffic on external port

**phase 2**

- $N^2$ internal links of capacity $R/N$
- each server receives up to $R$ bps traffic on external port

---
VLB: operation

- $N^2$ internal links of capacity $2R/N$
- each server receives up to $2R$ bps
- plus $R$ bps from external port
- hence, each server processes up to $3R$
- or up to $2R$, when traffic is uniform [directVLB, Liu’05]
Per-server fanout?

- Increase server capacity
- Add intermediate nodes
  - $k$-degree $n$-stage butterfly
Recap: cluster router solution

- Assign max external ports per server
- Full mesh, if possible
- Extra servers, otherwise
Takeaways

• Software routers provide extensibility

• Click modular router design
  – Add elements
  – Combine to create new functions via data flow

• RB
  – Smart single server optimizations
    • Multi-Q, core pinning etc
  – Incremental scaling with cluster router