RouteBricks: Exploiting Parallelism To Scale Software Routers

Mihai Dobrescu & Norbert Egi,
Katerina Argyraki, Byung-Gon Chun, Kevin Fall,
Gianluca Iannaccone, Allan Knies, Maziar Manesh,
Sylvia Ratnasamy

EPFL, Intel Labs Berkeley, Lancaster University

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

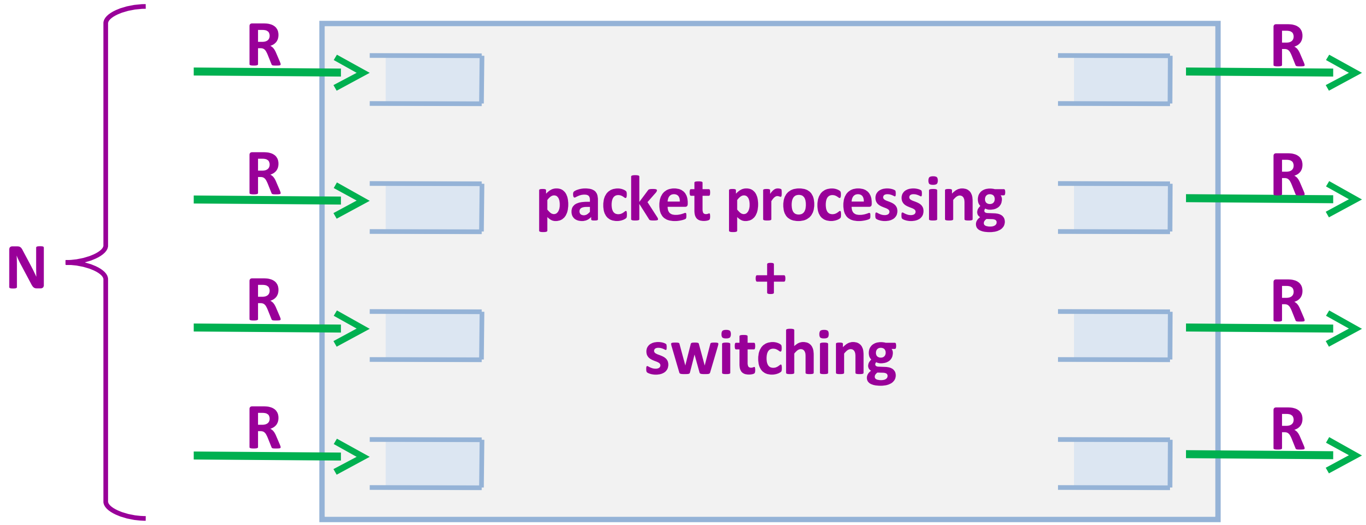
- **Fast “hardware” routers**
  - » throughput : Tbps
  - » no programmability
- **Programmable “software” routers**
  - » processing by general-purpose CPUs
  - » throughput < 10Gbps

RouteBricks

- **A router out of off-the-shelf PCs**
  - » familiar programming environment
  - » large-volume manufacturing

- ***Can we build a Tbps router out of PCs?***

Router =

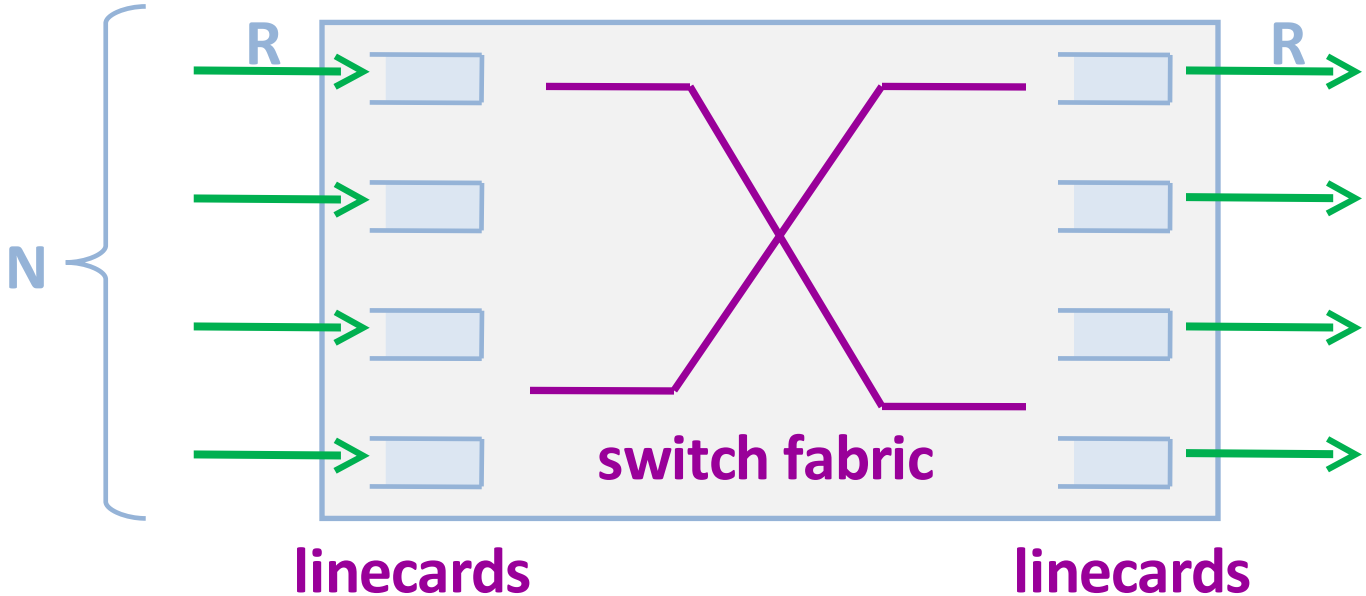


- *N*: number of external router ports
- *R*: external line rate

A hardware router

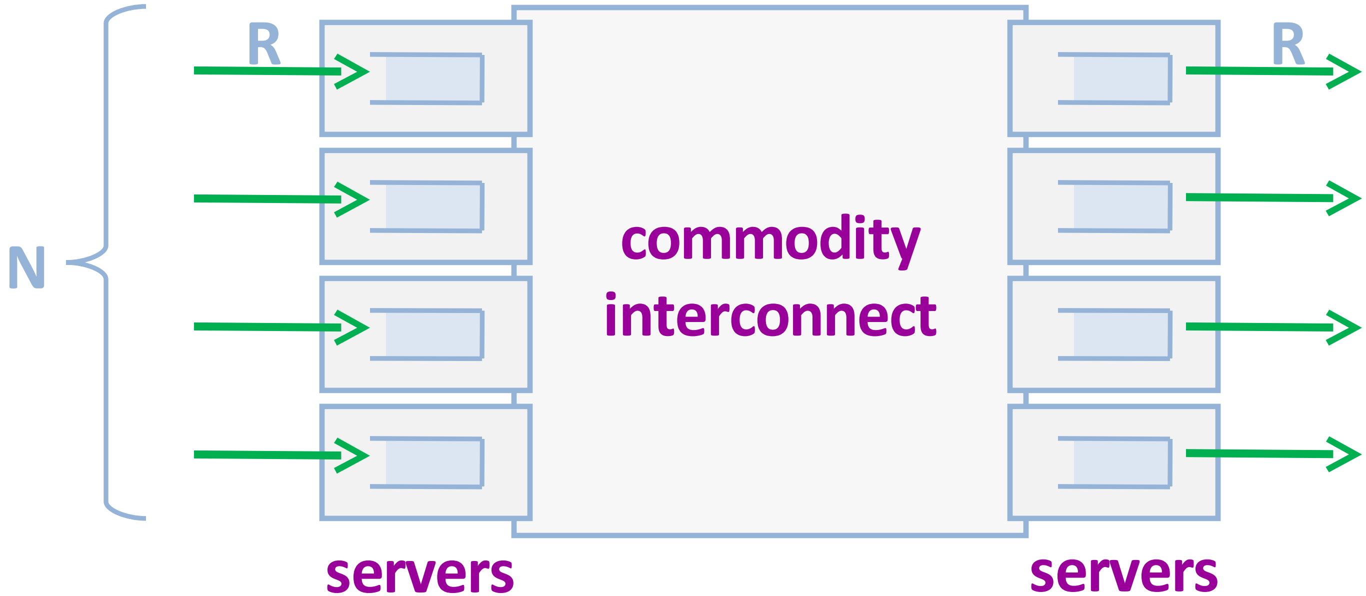
- Processing at rate ~*R* per linecard

A hardware router



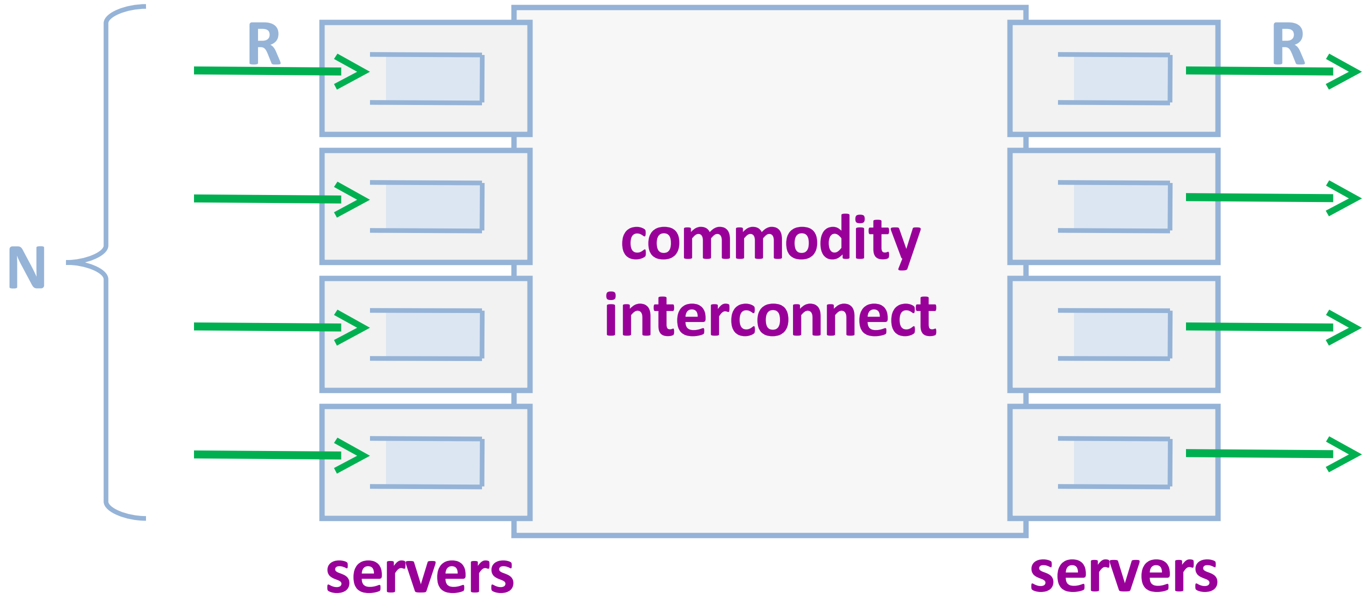
- Processing at rate ~*R* per linecard
- Switching at rate *N* x *R* by switch fabric

RouteBricks



- Processing at rate ~*R* per server
- Switching at rate ~*R* per server

RouteBricks



Per-server processing rate: c x R

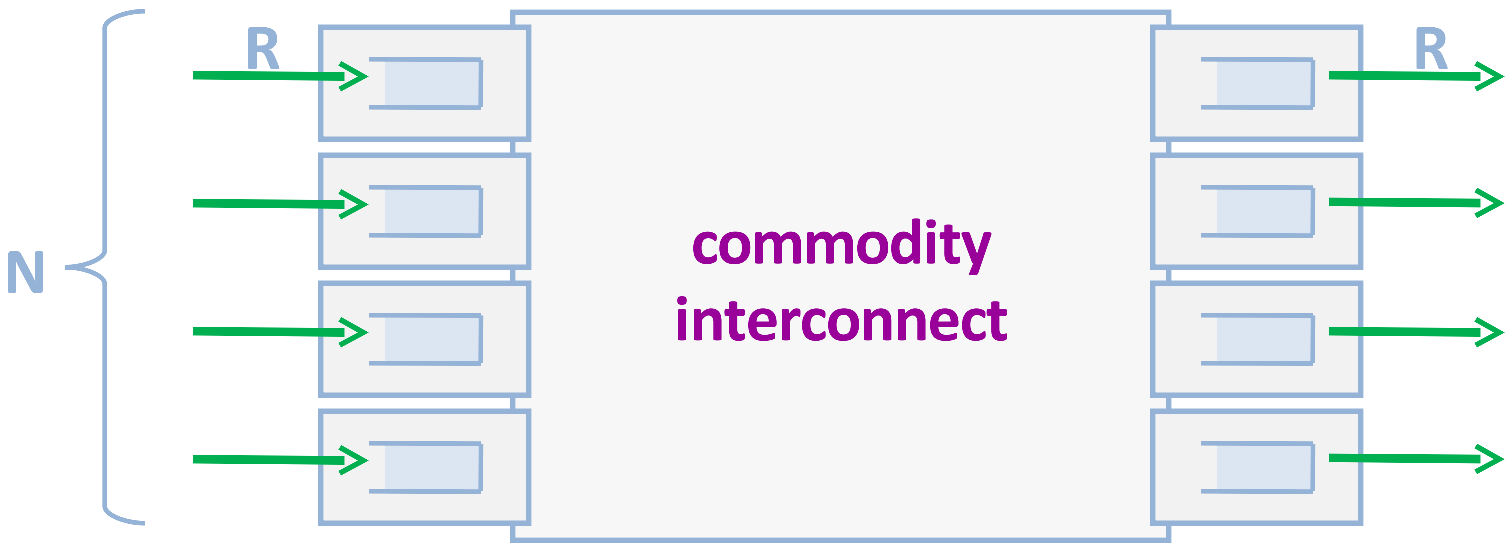
Outline

- Interconnect
- Server optimizations
- Performance
- Conclusions

Outline

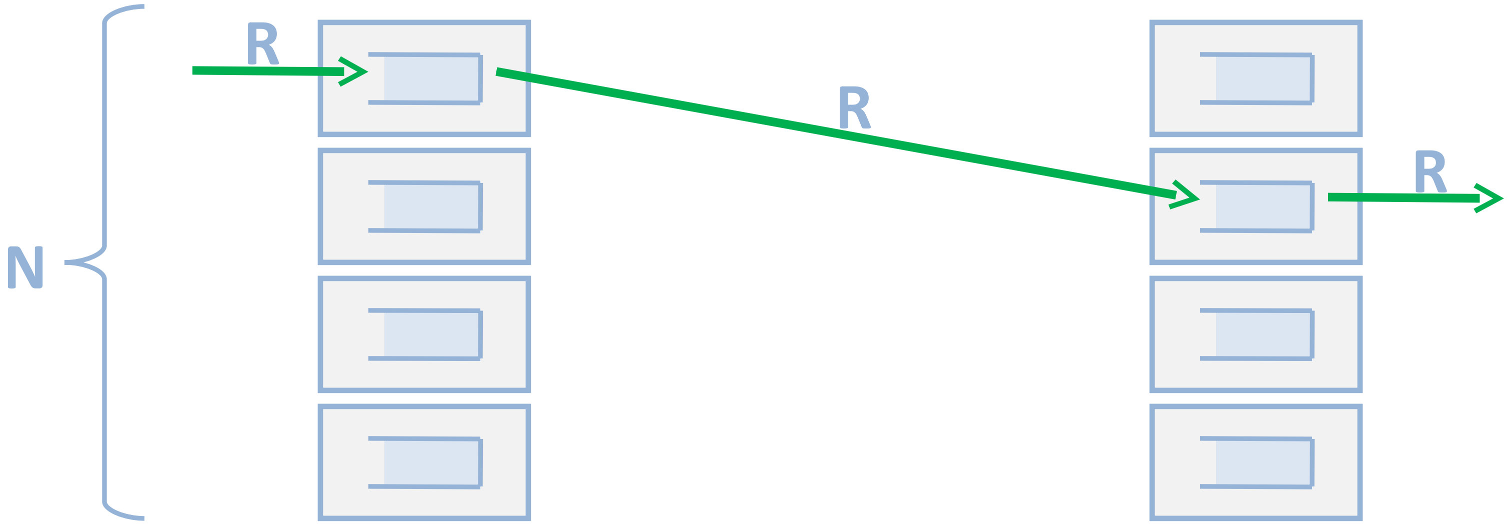
- Interconnect
- Server optimizations
- Performance
- Conclusions

Requirements

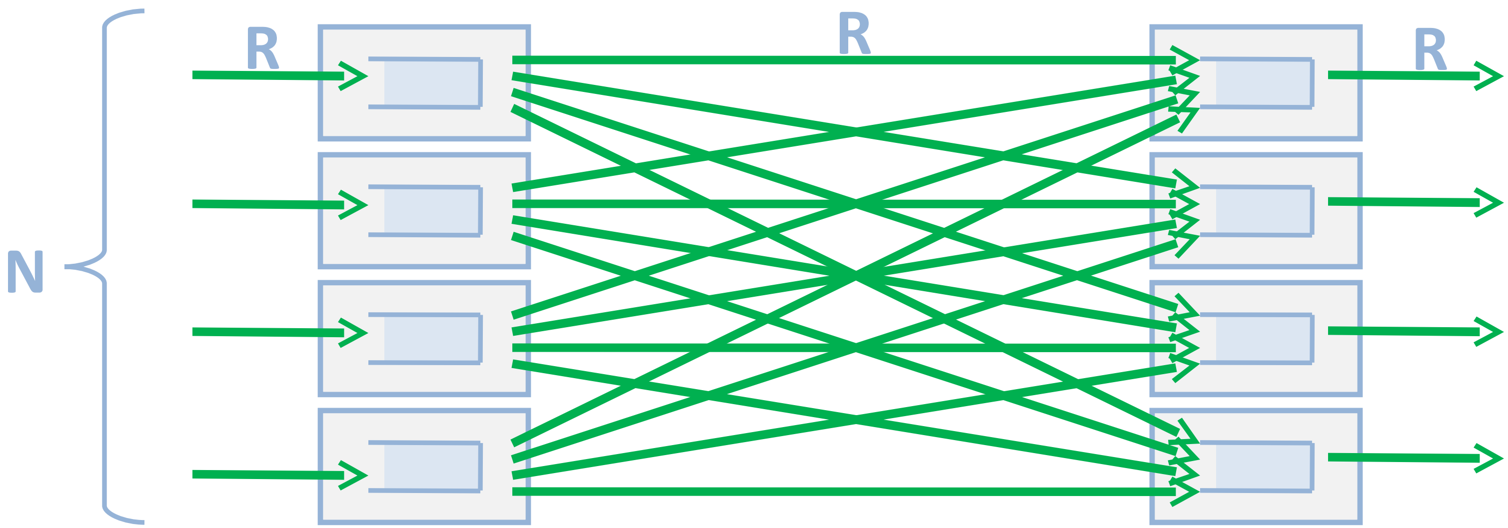


- Internal link rates < *R*
- Per-server processing rate: *c* x *R*
- Per-server fanout: constant

A naive solution

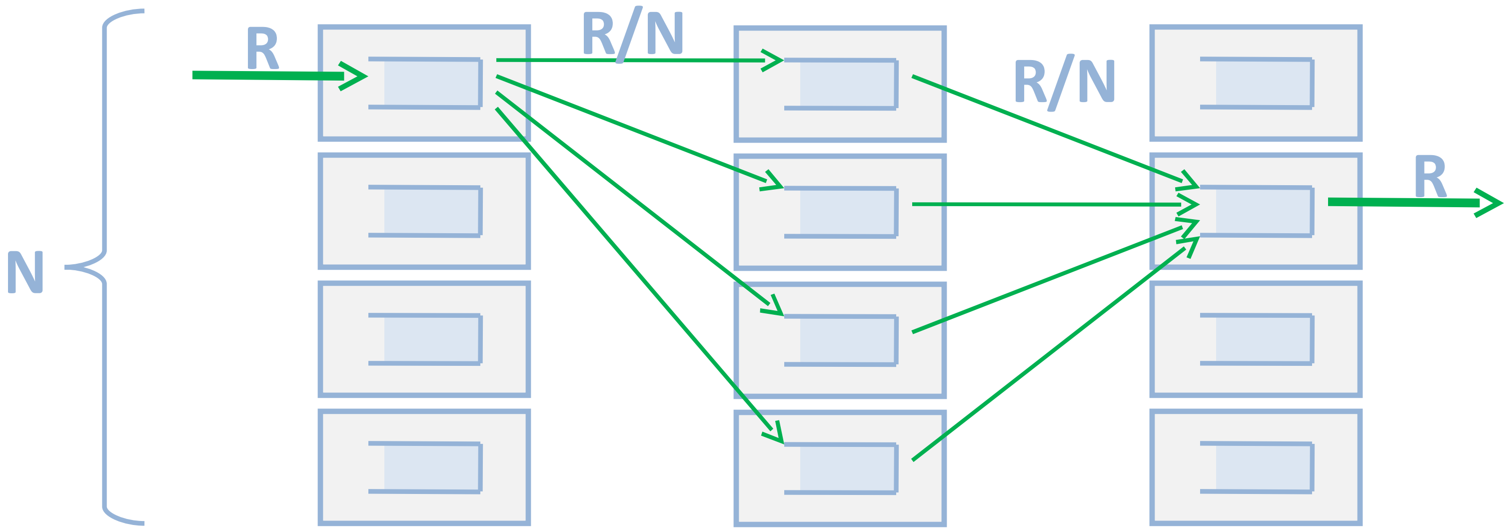


A naive solution

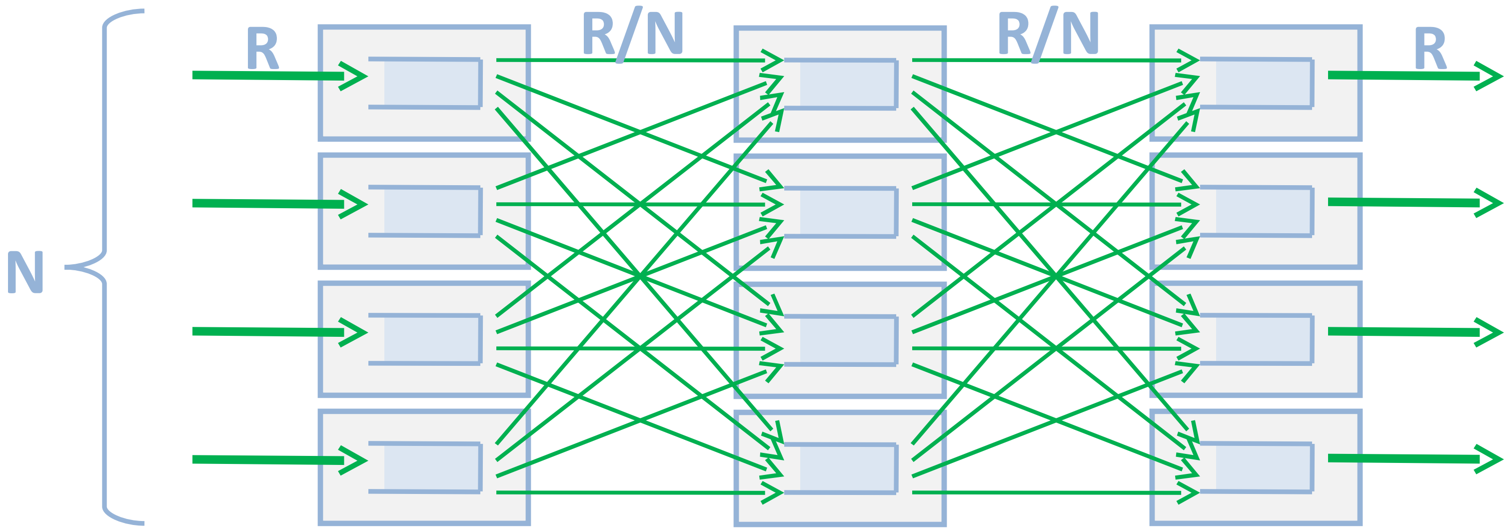


- *N* external links of capacity *R*
- *N*[2] internal links of capacity *R*

Valiant load balancing

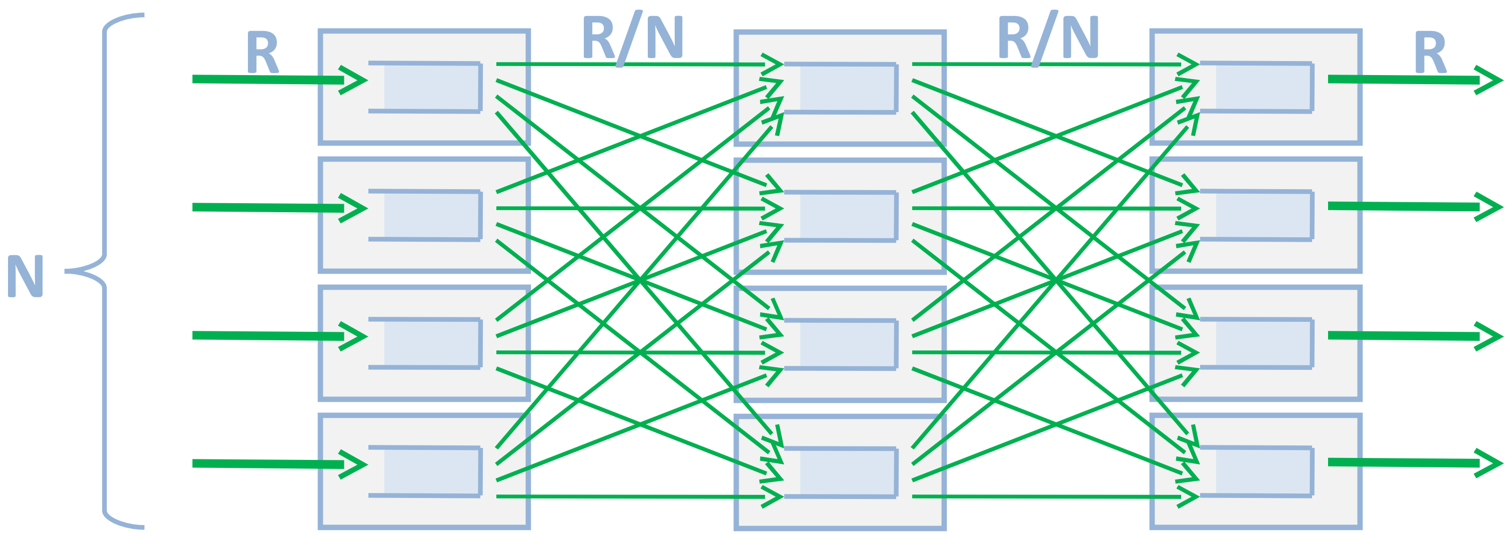


Valiant load balancing



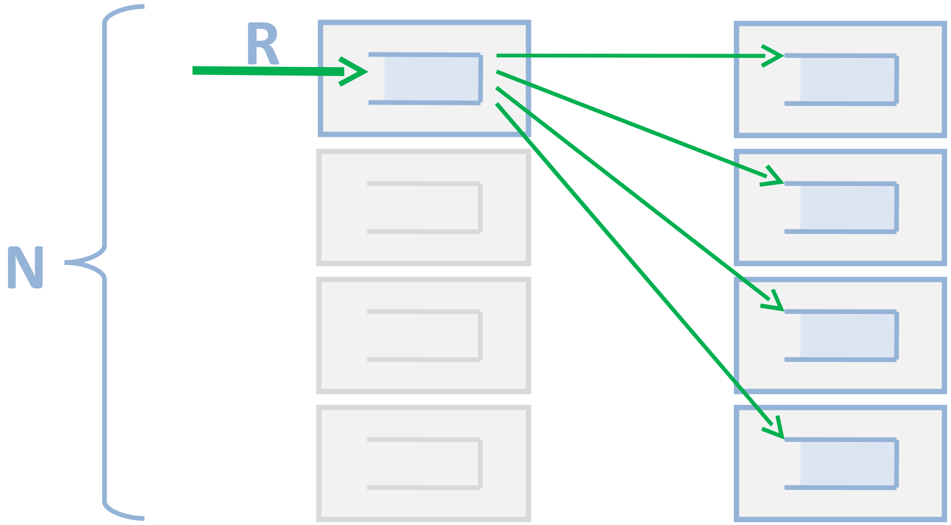
- *N* external links of capacity *R*
- *N*² internal links of capacity ~~*R*~~ ***2R/N***

Valiant load balancing

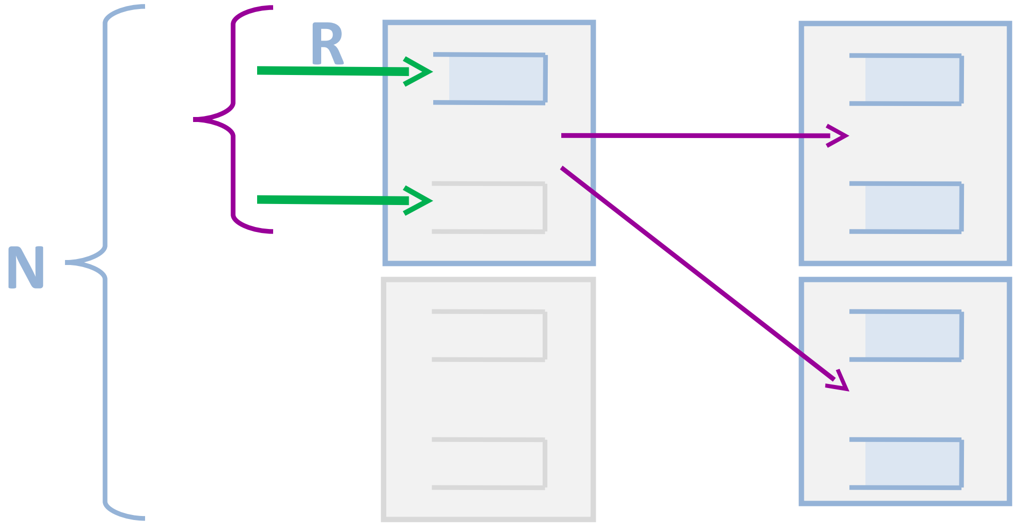


- Per-server processing rate: **3*R***
- Uniform traffic: **2*R***

Per-server fanout?

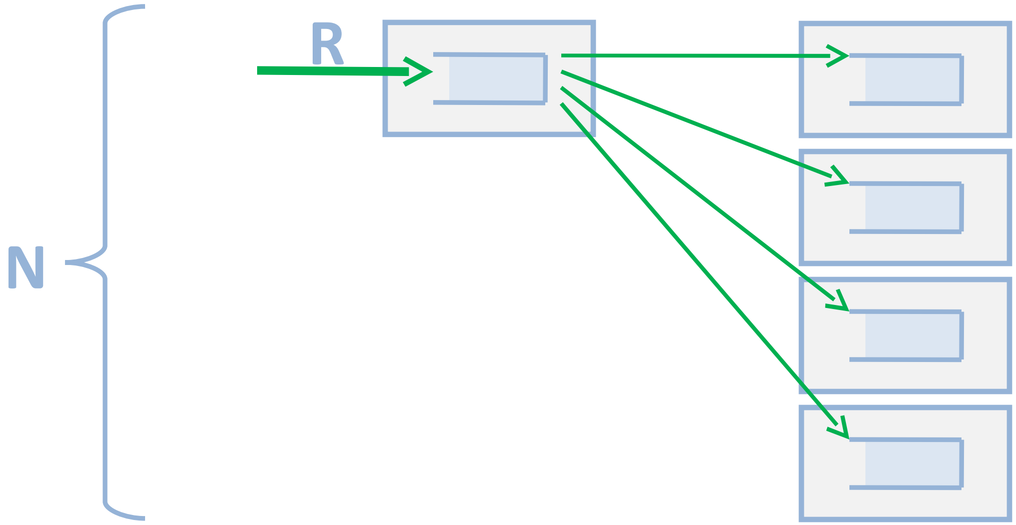


Per-server fanout?



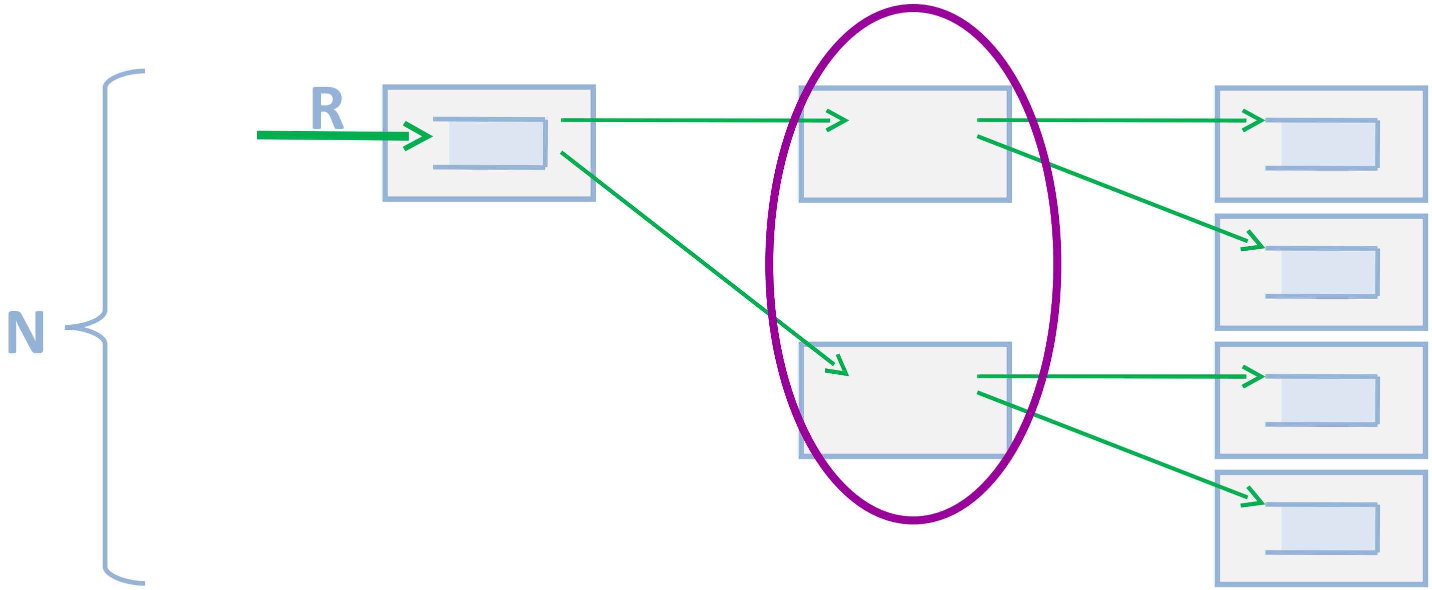
- Increase server capacity

Per-server fanout?



- Increase server capacity

Per-server fanout?



- **Increase server capacity**
- **Add intermediate nodes**
  - » *k*-degree *n*-stage butterfly

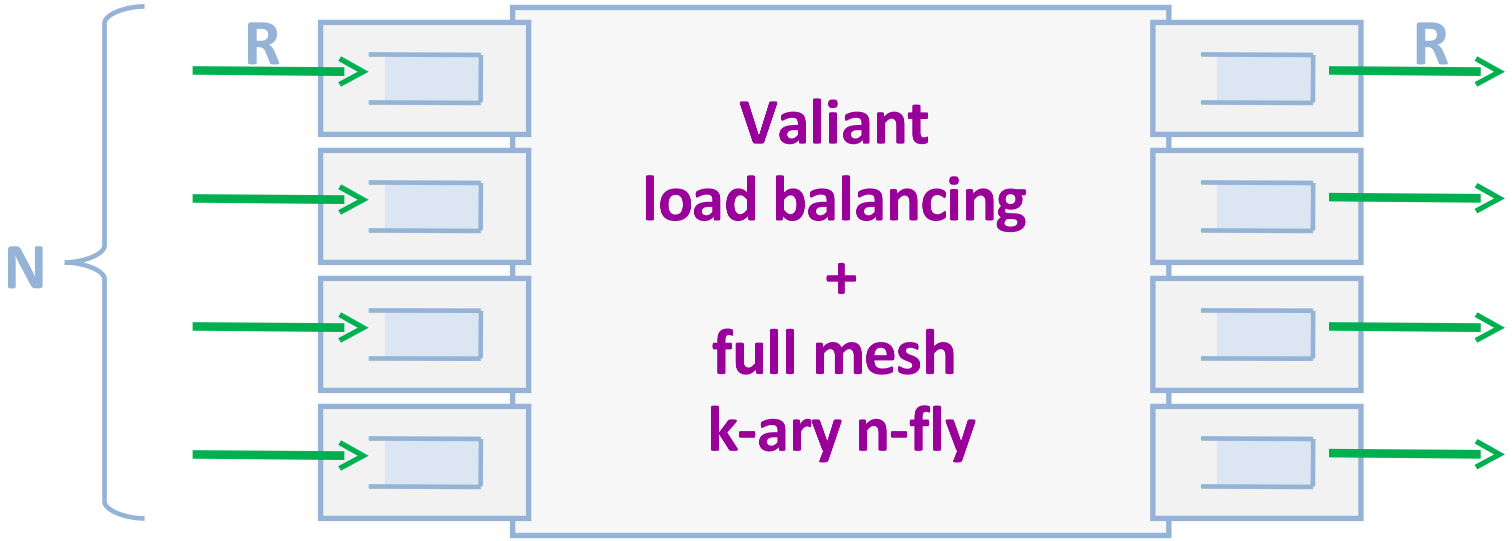
Our solution: combination

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

Example

- **Assuming current servers**
  - » 5 NICs, 2 x 10G ports or 8 x 1G ports
  - » 1 external port per server
- **N = 32 ports: full mesh**
  - » 32 servers
- **N = 1024 ports: 16-ary 4-fly**
  - » 2 extra servers per port

Recap

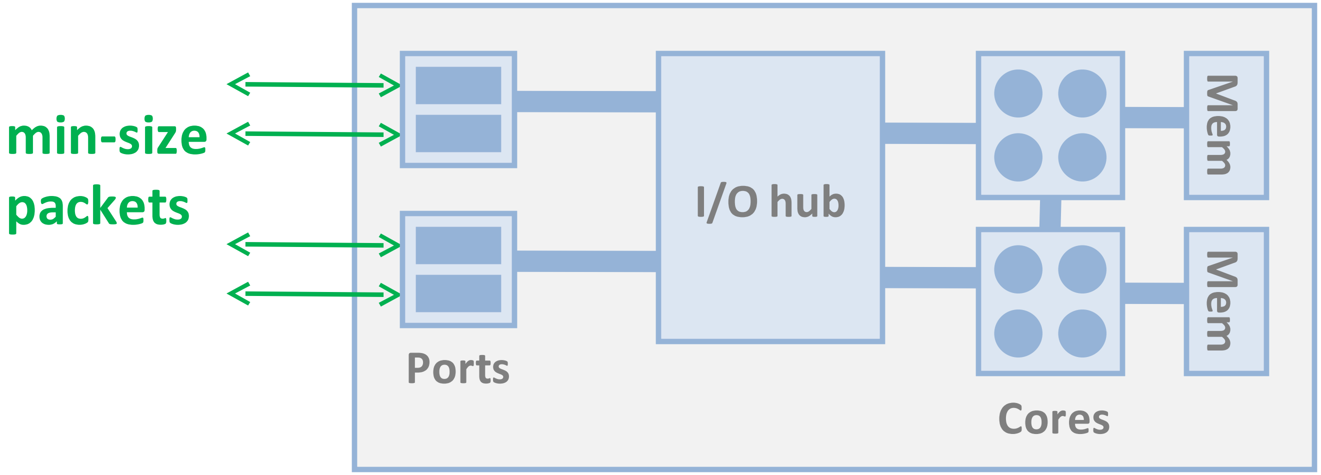


Per-server processing rate: 2*R* – 3*R*

Outline

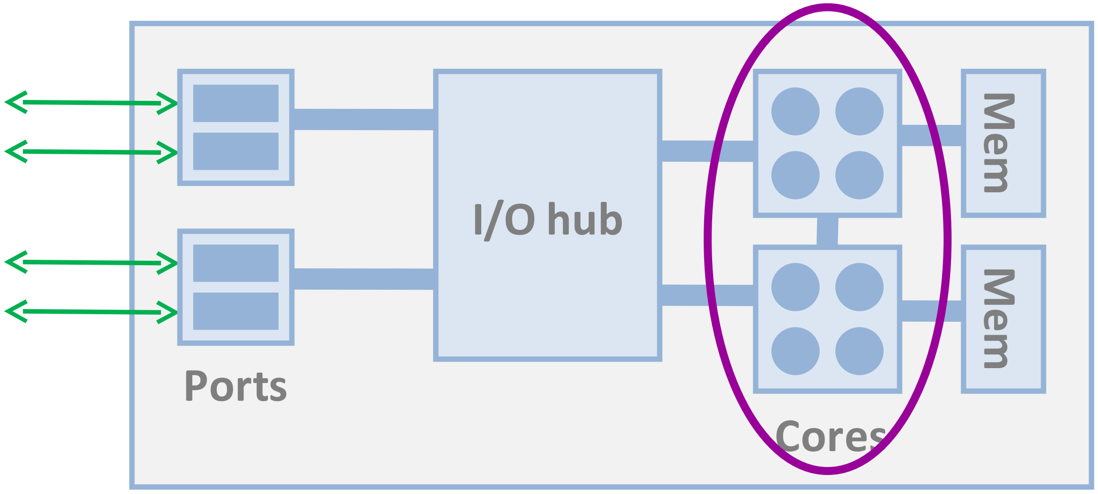
- Interconnect
- Server optimizations
- Performance
- Conclusions

Setup: NUMA architecture



- Nehalem architecture, QuickPath interconnect
- CPUs: 2 x [2.8GHz, 4 cores, 8MB L3 cache]
- NICs: 2 x Intel XFSR 2x10Gbps
- kernel-mode Click

Single-server performance

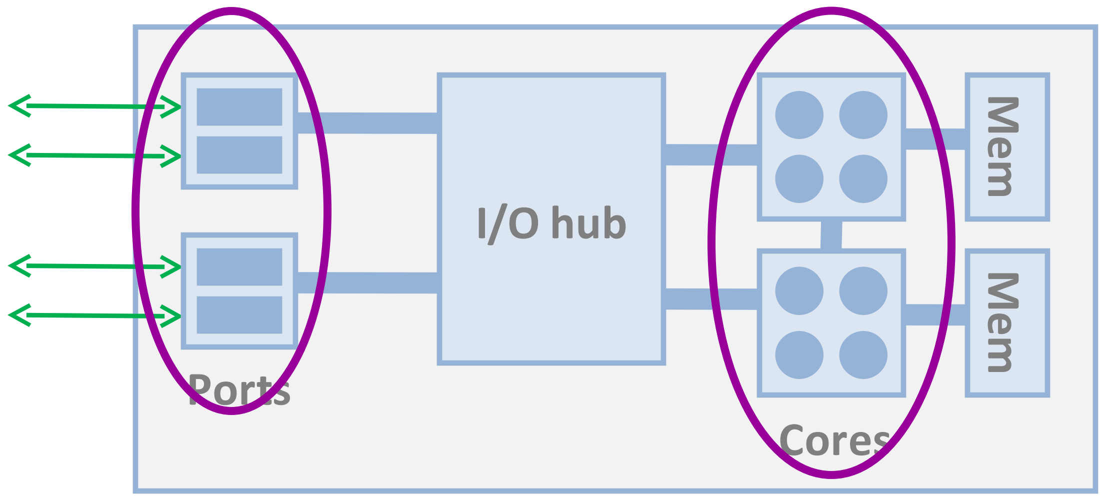


- First try: **1.3 Gbps**

Problem #1: book-keeping

- **Managing packet descriptors**
  - » moving between NIC and memory
  - » updating descriptor rings
- **Solution: batch packet operations**
  - » NIC batches multiple packet descriptors
  - » CPU polls for multiple packets

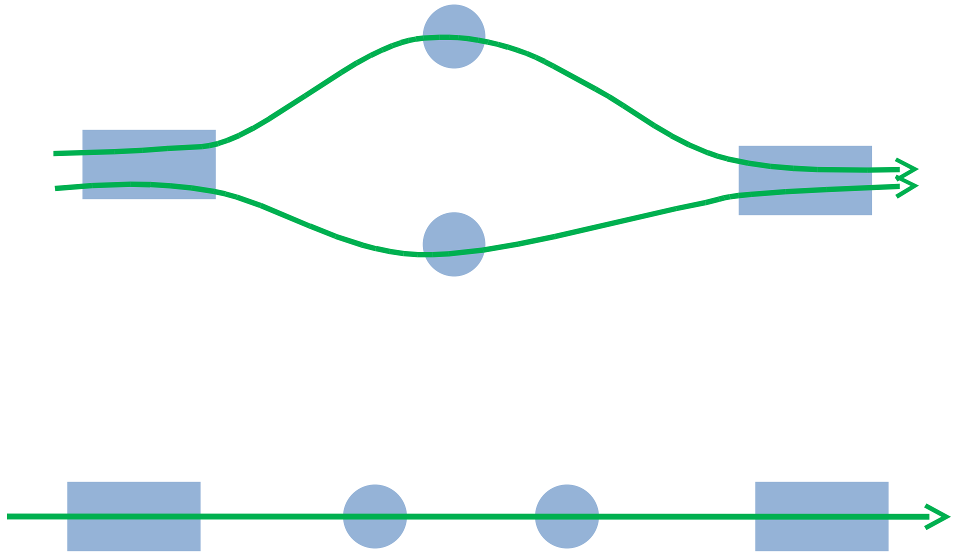
Single-server performance



- First try: 1.3 Gbps
- With batching: **3 Gbps**

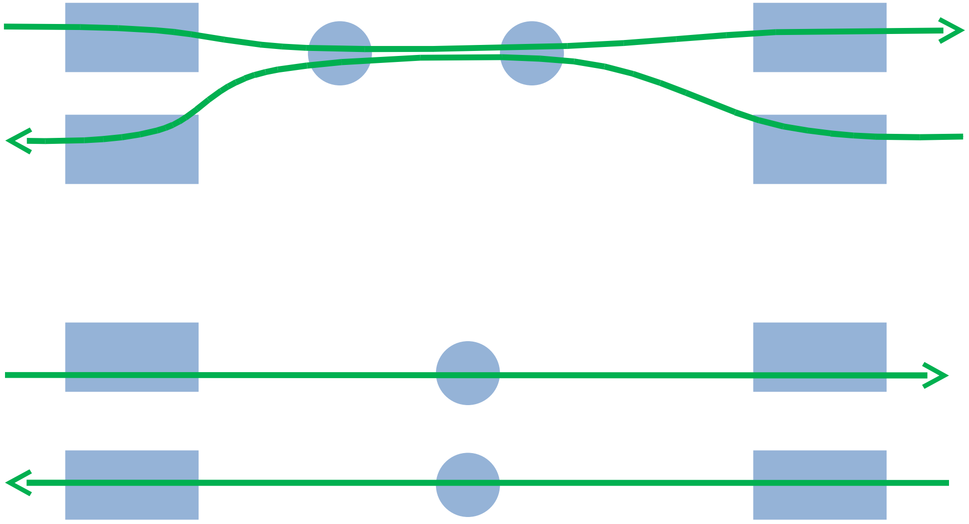
Problem #2: queue access

Problem #2: queue access



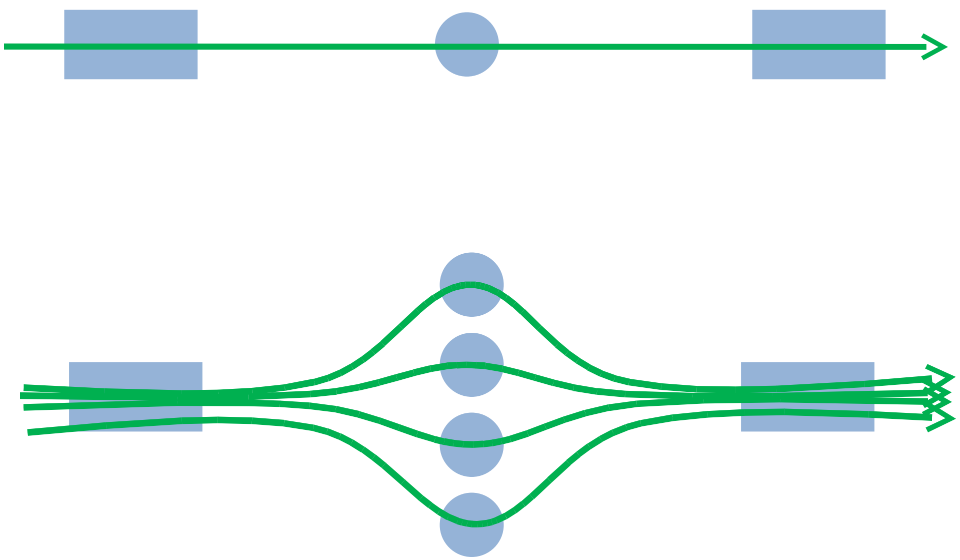
- **Rule #1: 1 core per port**

Problem #2: queue access



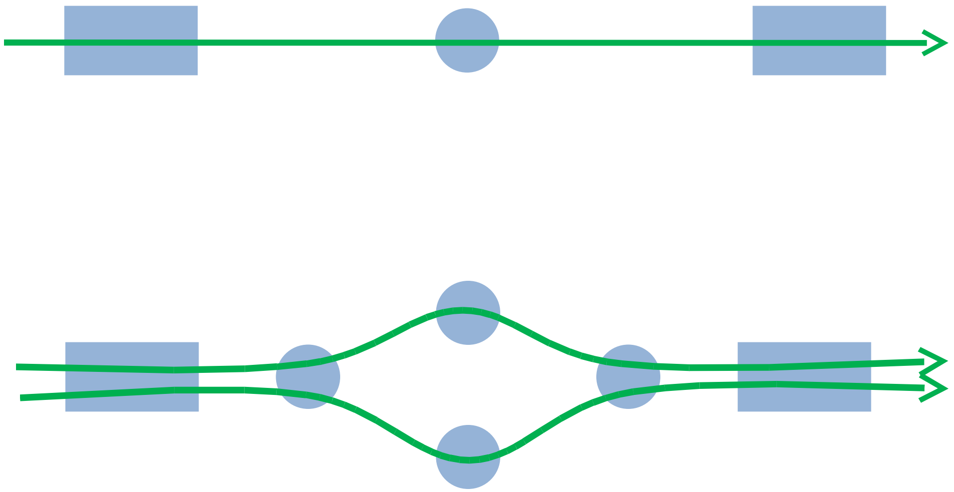
- Rule #1: 1 core per port
- Rule #2: 1 core per packet

Problem #2: queue access



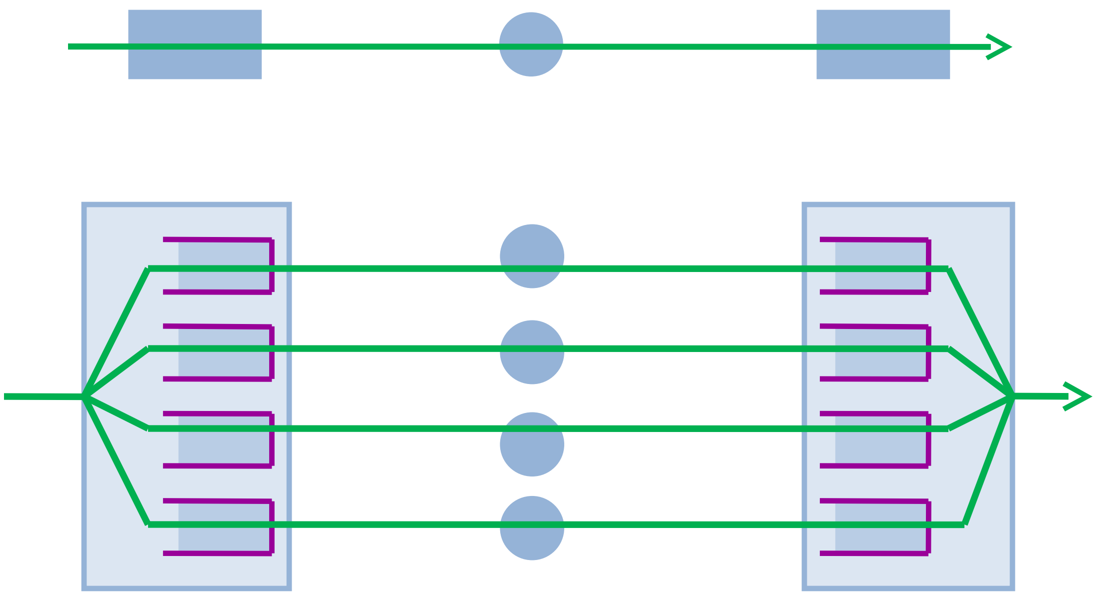
- Rule #1: 1 core per port
- Rule #2: 1 core per packet

Problem #2: queue access



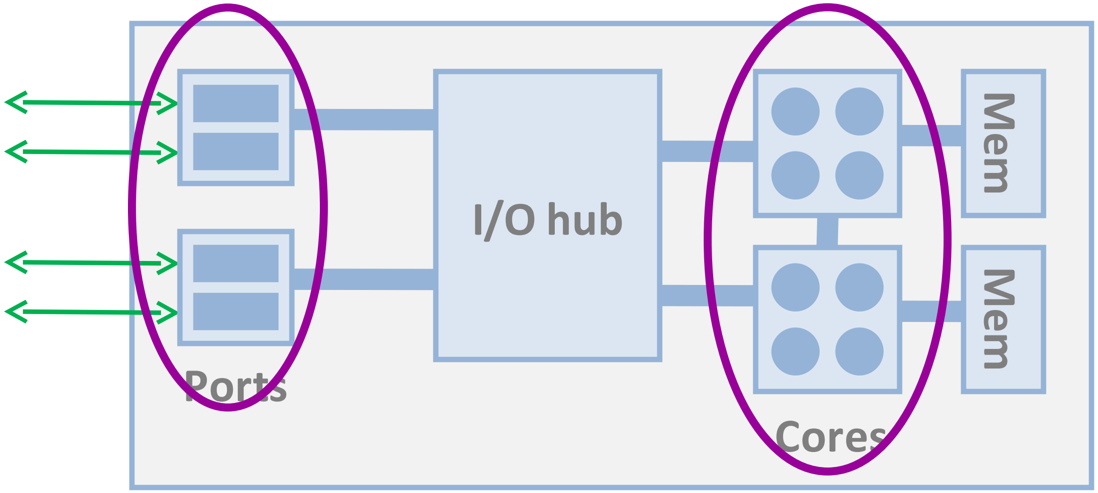
- **Rule #1: 1 core per port**
- **Rule #2: 1 core per packet** ✖

Problem #2: queue access



- Rule #1: 1 core per ~~port~~ queue
- Rule #2: 1 core per packet

Single-server performance



- First try: 1.3 Gbps
- With batching: 3 Gbps
- With multiple queues: 9.7 Gbps

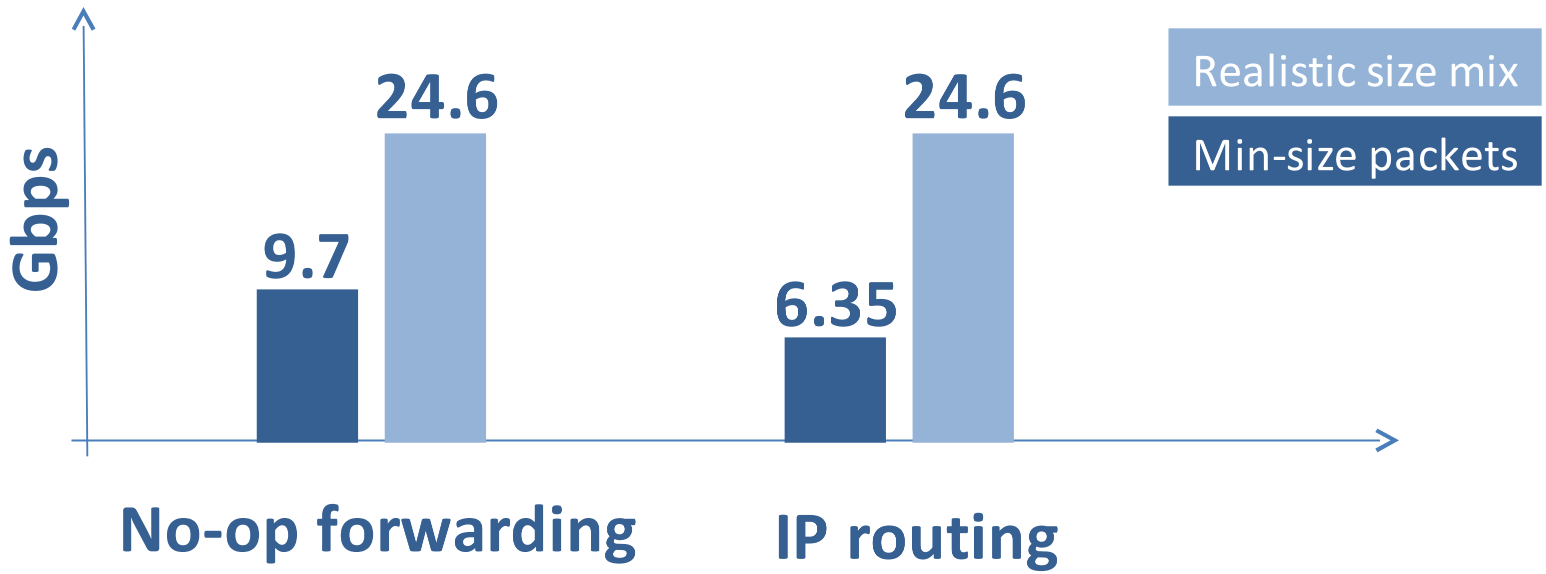
Recap

- **State-of-the art hardware**
  - » NUMA architecture, multi-queue NICs
- **Modified NIC driver**
  - » batching
- **Careful queue-to-core allocation**
  - » one core per queue, per packet

Outline

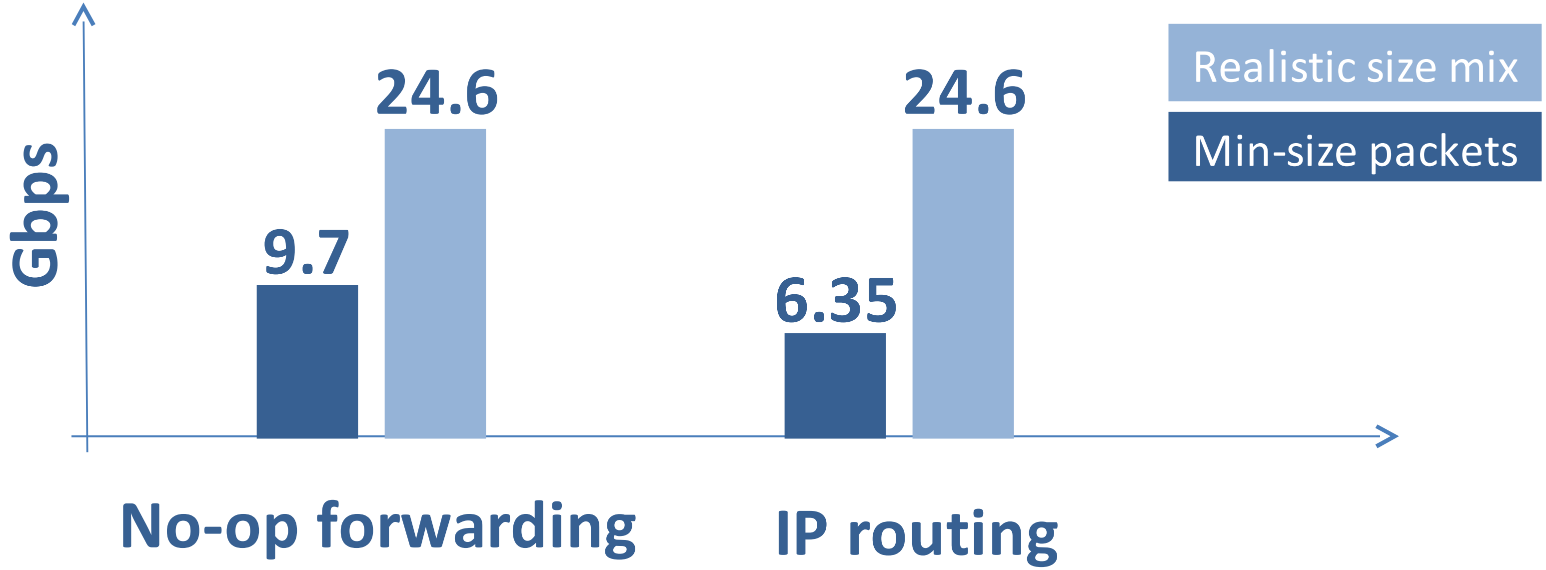
- Interconnect
- Server optimizations
- Performance
- Conclusions

Single-server performance



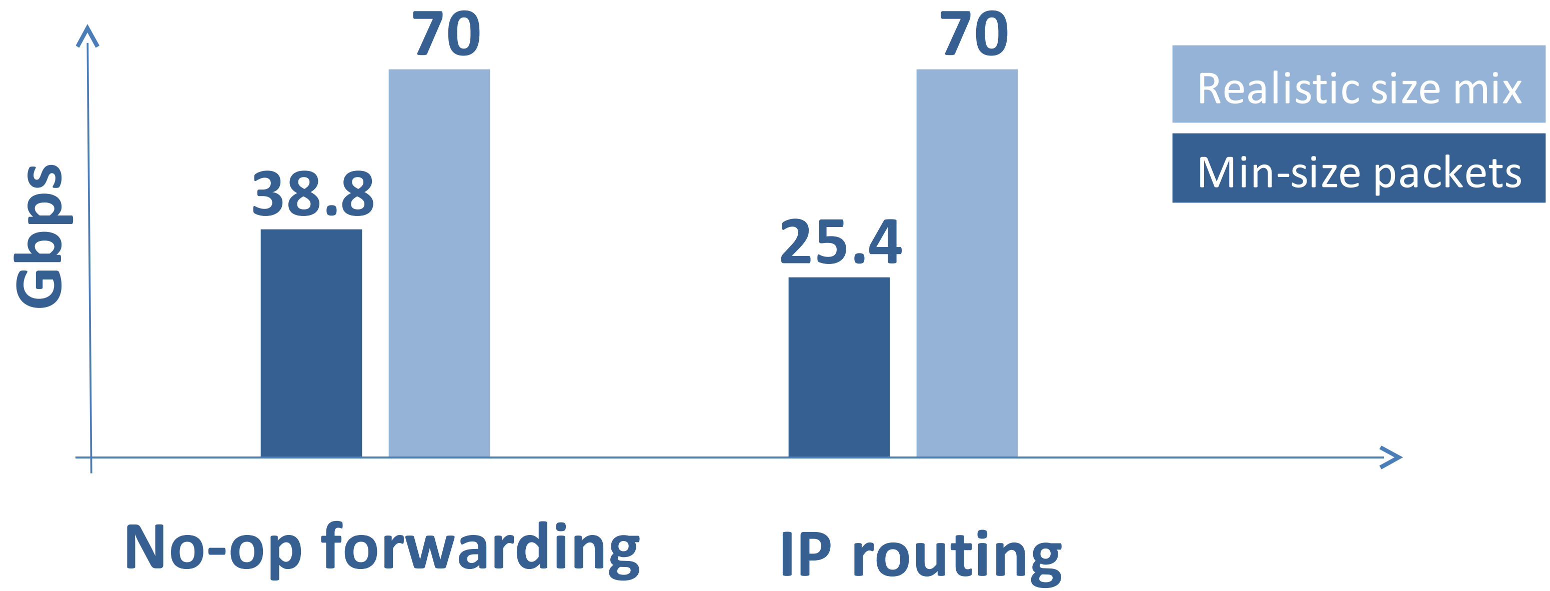
- **Realistic size mix: *R* = 8 – 12 Gbps**
- **Min-size packets: *R* = 2 – 3 Gbps**

Bottlenecks



- **Realistic size mix: I/O**
- **Min-size packets: CPU**

With upcoming servers



- Realistic size mix: *R* = 23 – 35 Gbps
- Min-size packets: *R* = 8.5 – 12.7 Gbps

RB4 prototype

- **N = 4 external ports**
  - 1 server per port
  - full mesh
- **Realistic size mix: 4 x 8.75 = 35 Gbps**
  - expected R = 8 – 12 Gbps
- **Min-size packets: 4 x 3 = 12 Gbps**
  - expected R = 2 – 3 Gbps

I did not talk about

- **Reordering**
  - » avoid per-flow reordering
  - » 0.15%
- **Latency**
  - » 24 microseconds per server (estimate)
- **Open issues**
  - » power, form-factor, programming model

Conclusions

- **RouteBricks: high-end software router**
  - » Valiant LB cluster of commodity servers
- **Programmable with Click**
- **Performance:**
  - » easily R = 1Gbps, N = 100s
  - » R = 10Gbps for realistic traffic
  - » for worst case, with upcoming servers

Thank you.

- NIC driver and more information at http://routebricks.org