EECS 262a
Advanced Topics in Computer Systems
Lecture 18

Software Routers/RouteBricks
March 30th, 2016

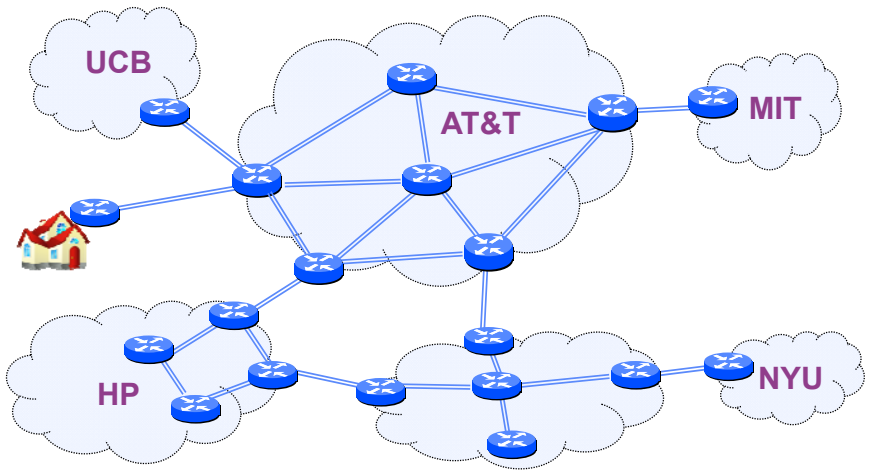
John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley
Slides Courtesy: Sylvia Ratnasamy

http://www.eecs.berkeley.edu/~kubitron/cs262

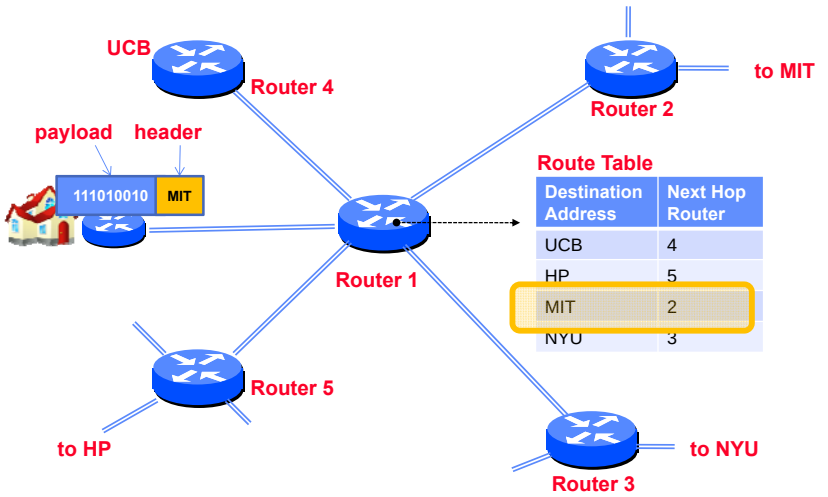
Today's Paper

- RouteBricks: Exploiting Parallelism To Scale Software Routers
  Mihai Dobrescu and Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall Gianluca Iannaccone, Allan Knies, Maziar Manesh, Sylvia Ratnasamy. Appears in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles* (SOSP), October 2009
- Thoughts?
- Paper divided into two pieces:
  - Single-Server Router
  - Cluster-Based Routing

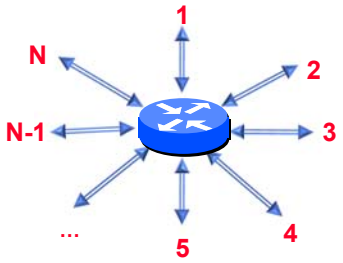
Networks and routers



Routers forward packets



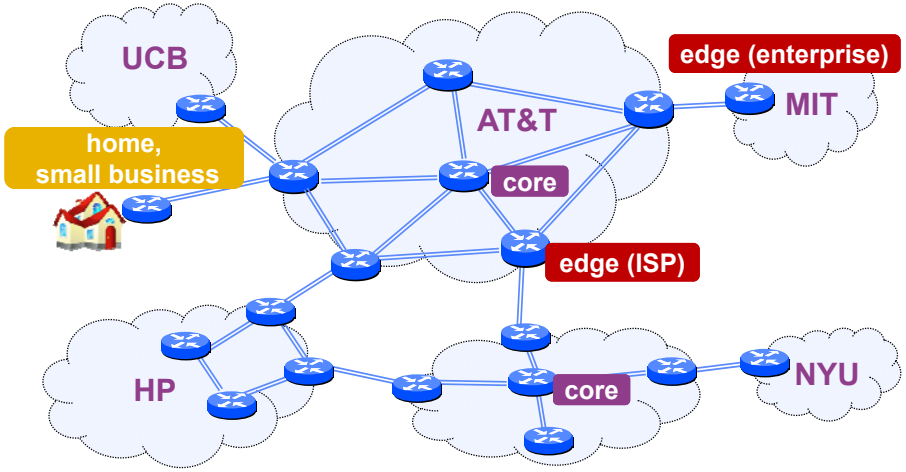
Router definitions



R *bits per second (bps)*

- N = number of external router `ports'
- R = line rate of a port
- Router capacity = N x R

Networks and routers



Examples of routers (core)

Juniper T640

- R= 2.5/10 Gbps
- NR = 320 Gbps

Cisco CRS-1

- **R=10/40 Gbps**
- **NR = 46 Tbps**

72 racks, 1MW

Examples of routers (edge)

Cisco ASR 1006

- **R=1/10 Gbps**
- **NR = 40 Gbps**

Juniper M120

- R= 2.5/10 Gbps
- NR = 120 Gbps

Examples of routers (small business)

Cisco 3945E

- R = 10/100/1000 Mbps
- NR < 10 Gbps

Building routers

- edge, core
  - ASICs
  - network processors
  - commodity servers ← RouteBricks
- home, small business
  - ASICs
  - network, embedded processors
  - commodity PCs, servers

Why programmable routers

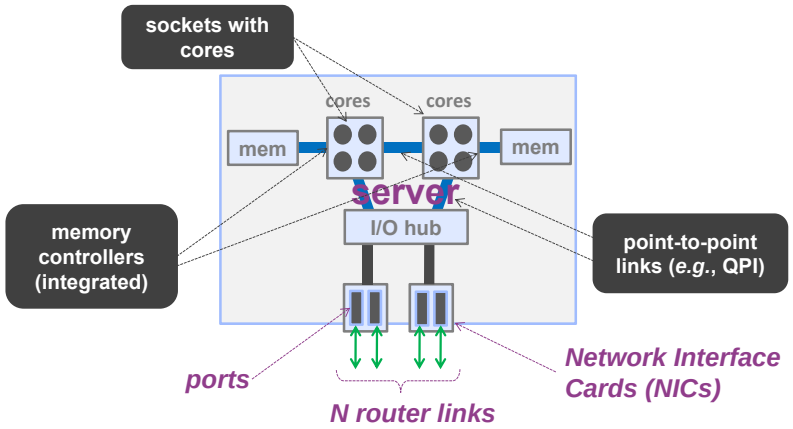
- New ISP services
  - intrusion detection, application acceleration
- Simpler network monitoring
  - measure link latency, track down traffic
- New protocols
  - IP traceback, Trajectory Sampling, …

Enable flexible, extensible networks

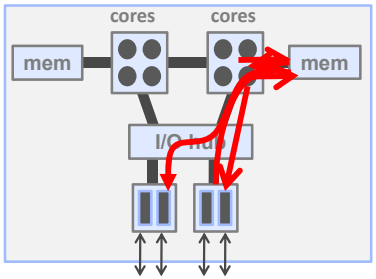
Challenge: performance

- deployed edge/core routers
  - port speed (R): 1/10/40 Gbps
  - capacity (NxR): 40Gbps to 40Tbps
- PC-based software routers
  - capacity (NxR), 2007: 1-2 Gbps [Click]
  - capacity (NxR), 2009: 4 Gbps [Vyatta]
- subsequent challenges: power, form-factor, …

A single-server router



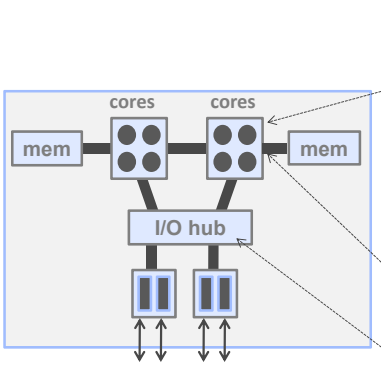
Packet processing in a server



Per packet,

1. core polls input port
2. NIC writes packet to memory
3. core reads packet
4. core processes packet (*address lookup, checksum, etc.*)
5. core writes packet to port

Packet processing in a server



8x 2.8GHz

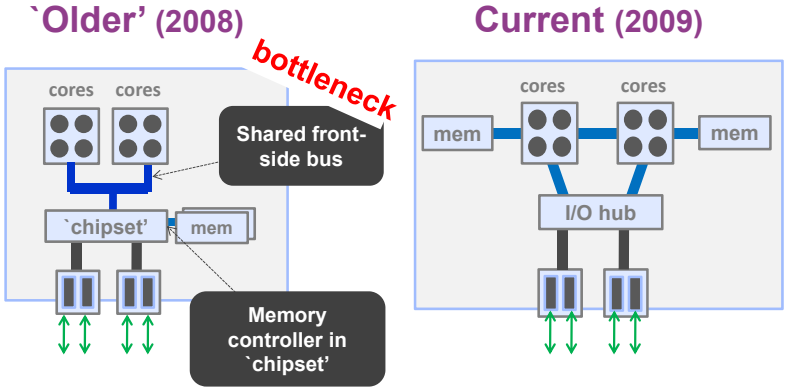
Assuming 10Gbps with all 64B packets
→19.5 million packets per second
→ one packet every 0.05 µsecs
→~1000 cycles to process a packet

Today, 200Gbps memory

Today, 144Gbps I/O

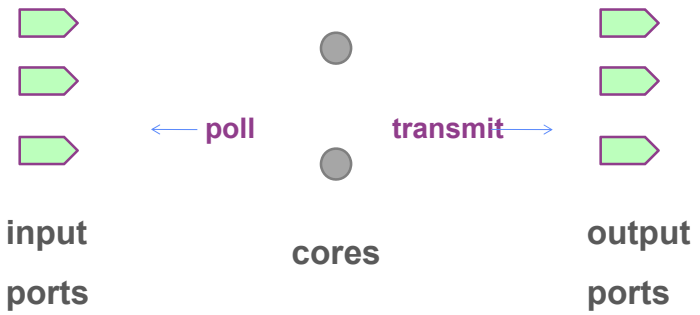
Suggests efficient use of CPU cycles is key!

Lesson#1: *multi-core alone isn't enough*



Hardware need: avoid shared-bus servers

Lesson#2: *on cores and ports*



How do we assign cores to input and output ports?

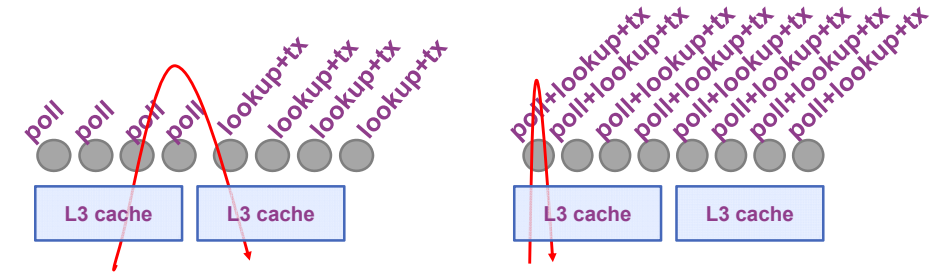
Lesson#2: *on cores and ports*

Problem: locking

Hence, rule: one core per port

Lesson#2: *on cores and ports*

Problem: cache misses, inter-core communication

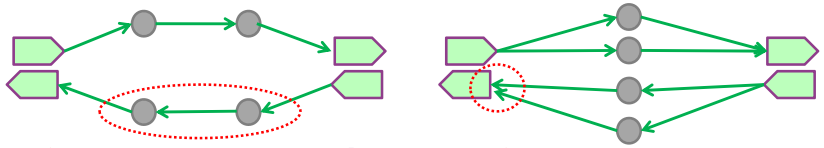


Hence, rule: one core per packet

Lesson#2: *on cores and ports*

- two rules:
  - one core per port
  - one core per packet
- problem: often, can't simultaneously satisfy both

Example: when #cores > #ports

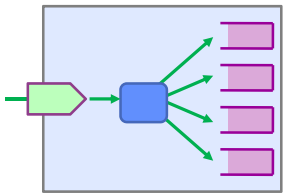


✖ *one core per packet* ✖ *one core per port*

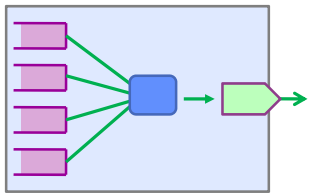
- solution: use *multi-Q* NICs

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)



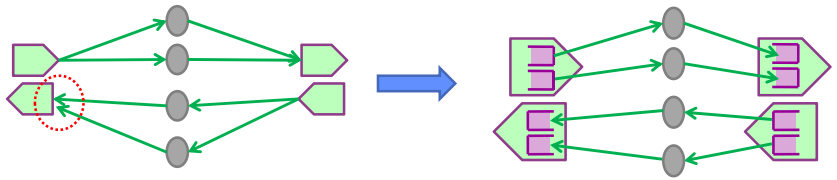
Multi-Q NIC: incoming traffic



Multi-Q NIC: outgoing traffic

Multi-Q NICs

- feature on modern NICs (for virtualization)
- repurposed for routing
  - rule: one core per port
  - rule: one core per ~~packet~~ queue



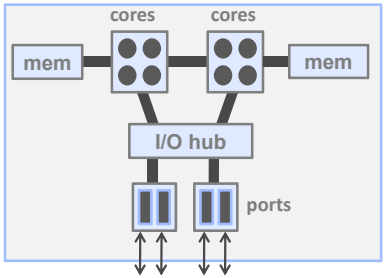
- if #queues per port == #cores, can always enforce both rules

Lesson#2: *on cores and ports*

recap:

- use multi-Q NICs
  - with modified NIC driver for lock-free polling of queues
- with
  - one core per queue (avoid locking)
  - one core per packet (avoid cache misses, inter-core communication)

Lesson#3: book-keeping



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

problem: excessive per packet book-keeping overhead

- solution: batch packet operations
  - NIC transfers packets in batches of `k'

Recap: routing on a server

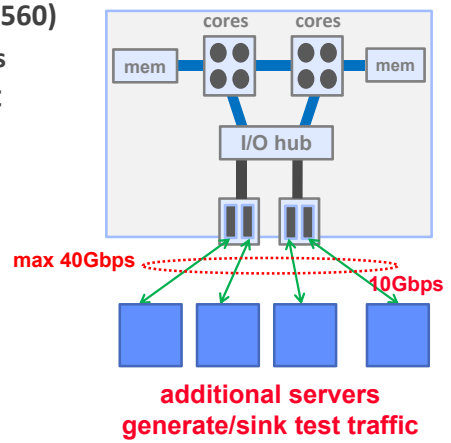
Design lessons:

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**

(see paper for "non needs" – careful memory placement, etc.)

Single-Server Measurements: Experimental setup

- **test server: Intel Nehalem (X5560)**
  - **dual socket, 8x 2.80GHz cores**
  - **2x NICs; 2x 10Gbps ports/NIC**



Project Feedback from Meetings

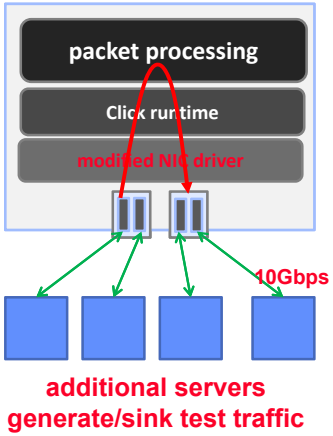
- Should have Updated your project descriptions and plan
  - Turn your description/plan into a living document in Google Docs
  - Share Google Docs link with us
  - Update plan/progress throughout the semester
- Questions to address:
  - What is your evaluation methodology?
  - What will you compare/evaluate against? Strawman?
  - What are your evaluation metrics?
  - What is your typical workload? Trace-based, analytical, …
  - Create a concrete staged project execution plan:
    - » Set reasonable initial goals with incremental milestones – always have something to show/results for project

Midterm: Over Weekend?

- **Out Wednesday**
- **Due 11:59PM PST a week from Tomorrow (11/11)**
- **Rules:**
  - **Open book**
  - **No collaboration with other students**

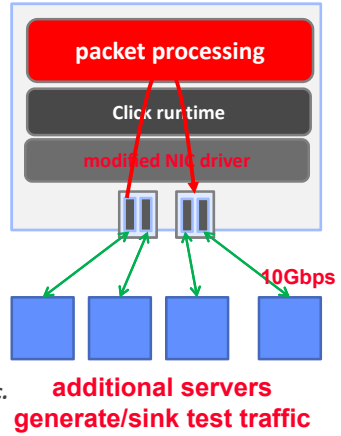
Experimental setup

- test server: Intel Nehalem (X5560)
- software: kernel-mode Click [TOCS'00]
  - with modified NIC driver (batching, multi-Q)



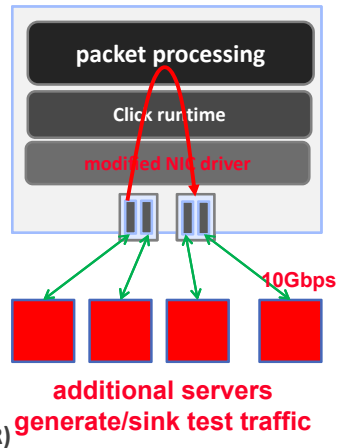
Experimental setup

- test server: Intel Nehalem (X5560)
- software: kernel-mode Click [TOCS'00]
  - with modified NIC driver
- packet processing
  - static forwarding (no header processing)
  - IP routing
    - » *trie-based longest-prefix address lookup*
    - » *~300,000 table entries [RouteViews]*
    - » *checksum calculation, header updates, etc.*

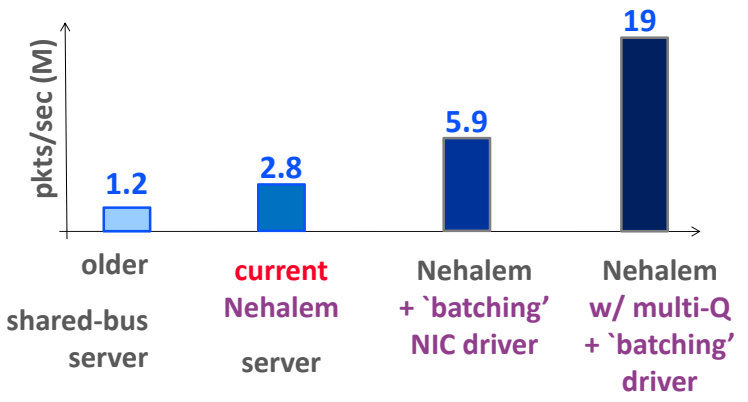


Experimental setup

- test server: Intel Nehalem (X5560)
- software: kernel-mode Click [TOCS'00]
  - with modified NIC driver
- packet processing
  - static forwarding (no header processing)
  - IP routing
- input traffic
  - all min-size (64B) packets (maximizes packet rate given port speed R)
  - realistic mix of packet sizes [Abilene]

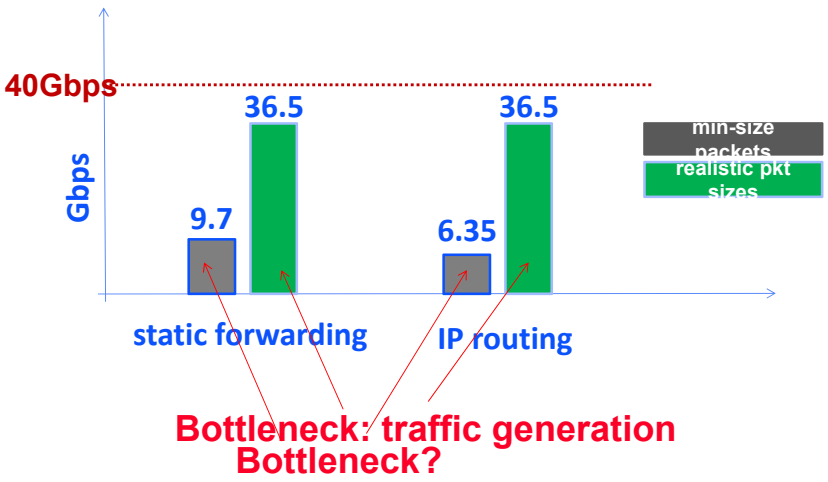


Factor analysis: design lessons



Test scenario: static forwarding of min-sized packets

Single-server performance



Bottleneck analysis (64B pkts)

Recall: max IP routing = 6.35Gbps → 12.4 M pkts/sec

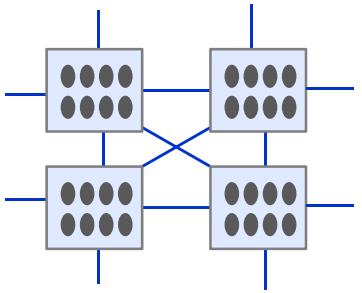
	Per-packet load due to routing	Maximum component capacity – nominal (empirical)	Max. packet rate as per component capacity -- nominal (empirical)
memory			
I/O		CPUs are the bottleneck	
Inter-socket link			
CPUs	**1693** cycles/pkt	**22.4** Gcycles/sec	**13** Mpkts/sec

Test scenario: IP routing of min-sized packets

Recap: single-server performance

	R	NR
current servers (realistic packet sizes)	1/10 Gbps	36.5 Gbps
current servers (min-sized packets)	1	6.35 (CPUs bottleneck)

Recap: single-server performance



With upcoming servers? (2010)

4x cores, 2x memory, 2x I/O

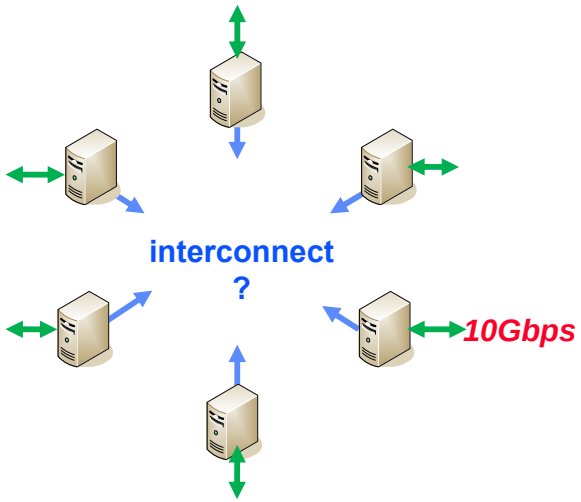
Recap: single-server performance

	R	NR
current servers (realistic packet sizes)	1/10 Gbps	36.5 Gbps
current servers (min-sized packets)	1	6.35 (CPUs bottleneck)
upcoming servers – estimated (realistic packet sizes)	1/10/40	146
upcoming servers – estimated (min-sized packets)	1/10	25.4

Practical Architecture: Goal

- scale software routers to multiple 10Gbps ports
- example: 320Gbps (32x 10Gbps ports)
  - higher-end of edge routers; lower-end core routers

A cluster-based router today

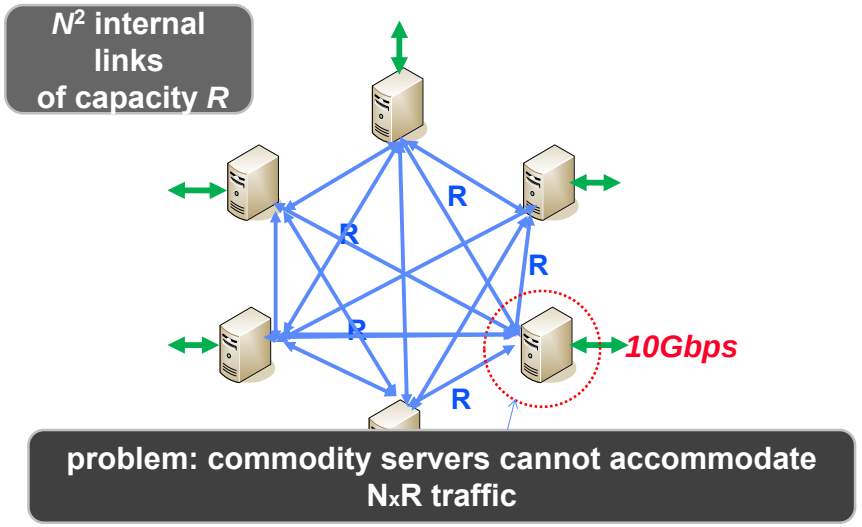


Interconnecting servers

Challenges

- any input can send up to R *bps* to any output

A naïve solution

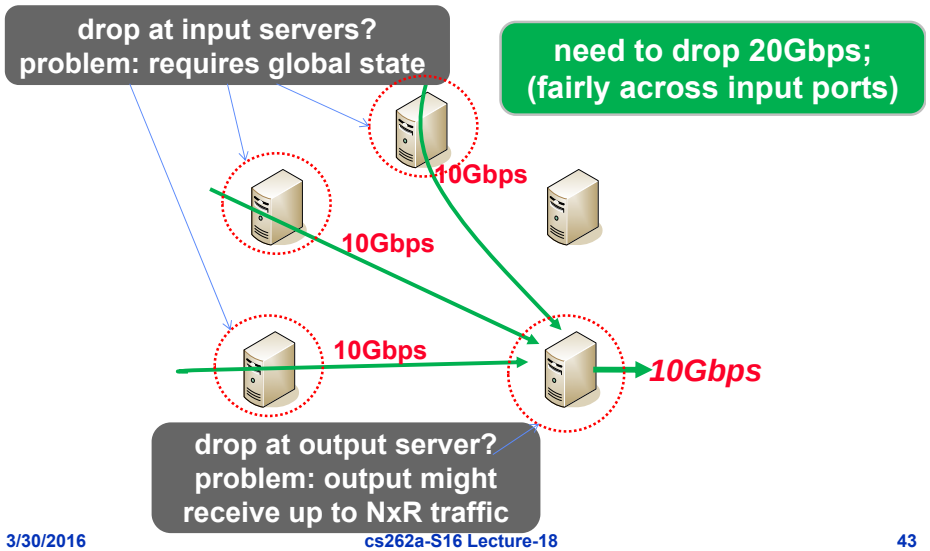


Interconnecting servers

Challenges

- any input can send up to R *bps* to any output
  - » but need a low-capacity interconnect (~NR)
  - » *i.e.,* fewer (<N), lower-capacity (<R) links per server
- must cope with overload

Overload



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)

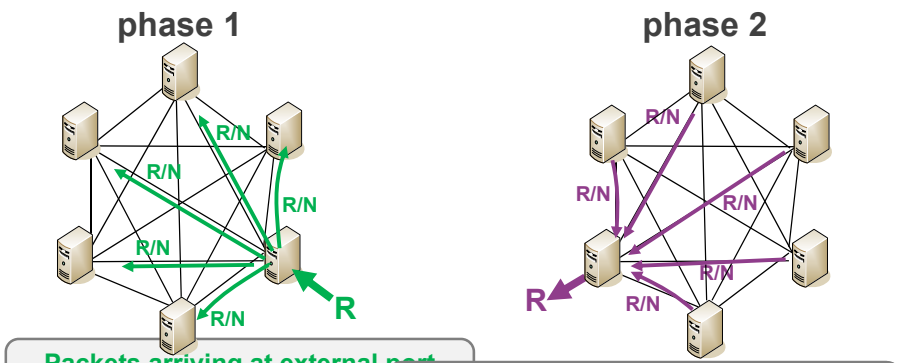
- internal link rates ≤ *R*
- per-node processing: $c_x R$ *(small c)*
- limited per-node fanout

Solution: Use Valiant Load Balancing (VLB)

Valiant Load Balancing (VLB)

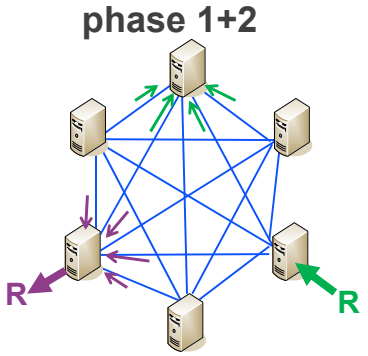
- Valiant *et al.* [STOC'81], communication in multi-processors
- applied to data centers [Greenberg'09], all-optical routers [Kesslassy'03], traffic engineering [Zhang-Shen'04], *etc.*
- idea: random load-balancing across a low-capacity interconnect

VLB: operation



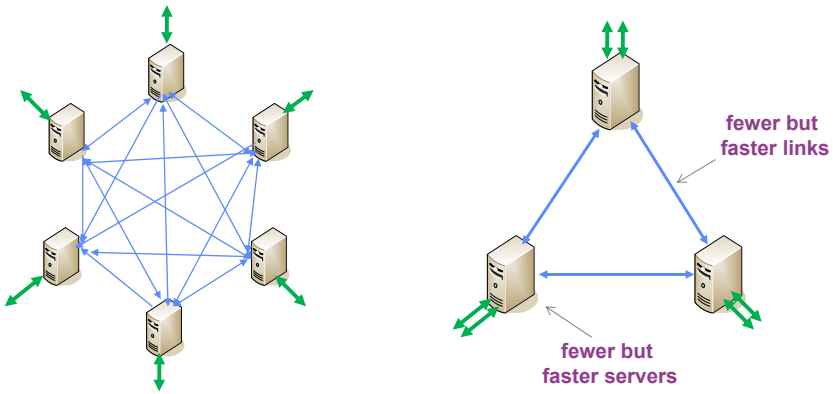
Packets forwarded in two phases

VLB: operation



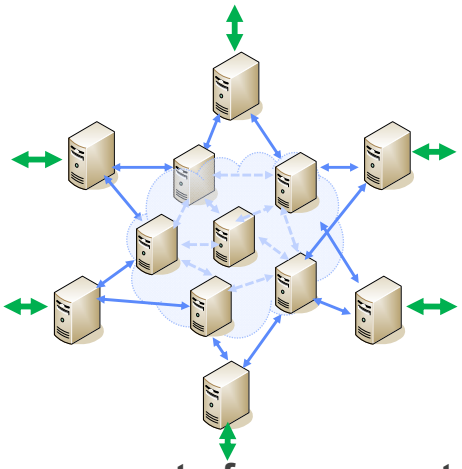
- N^2 internal links of capacity 2R/N
- each server receives up to 2R *bps*
- plus R bps from external port
- hence, each server processes up to 3R
- or up to 2R, when traffic is uniform [directVLB, Liu'05]

VLB: fanout? (1)



Multiple external ports per server
(if server constraints permit)

VLB: fanout? (2)



Use extra servers to form a constant-degree multi-stage interconnect (*e.g.*, butterfly)

Authors solution:

- assign maximum external ports per server
- servers interconnected with commodity NIC links
- servers interconnected in a full mesh if possible
- else, introduce extra servers in a k-degree butterfly
- servers run flowlet-based VLB

Scalability

- question: how well does clustering scale for realistic server fanout and processing capacity?
- metric: number of servers required to achieve a target router speed

Scalability

Assumptions

- 7 NICs per server
- each NIC has 6 x 10Gbps ports *or* 8 x 1Gbps ports
- current servers
  - **one** external 10Gbps port per server
    (i.e., requires that a server process 20-30Gbps)
- upcoming servers
  - **two** external 10Gbps port per server
    (i.e., requires that a server process 40-60Gbps)

Example: 320Gbps

- R=10Gbps, N=32
- with current servers: 1x 10Gbps external port
  - target: 32 servers
  - 2R/N < 1Gbps → need: 1Gbps internal links
  - 8x 1Gbps ports/NIC → need: 4 NICs per server

Example: 320Gbps

- R=10Gbps, N=32
- with current servers: 1x 10Gbps external port
  - need: 32 servers, 4 NICs/server (1Gbps NICs)

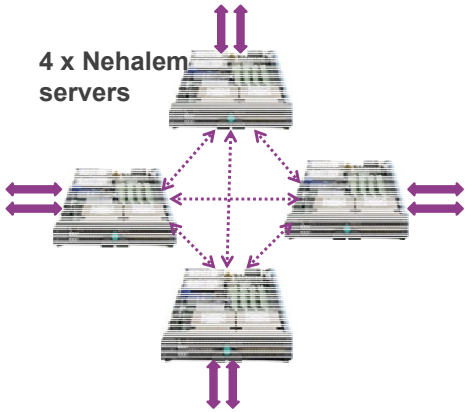
Scalability (computed)

	160Gbps	320Gbps	640Gbps	1.28Tbps	2.56Tbps
current servers	16	32 →	128	256	512
upcoming servers	8	16	32 →	128	256

Transition from mesh to butterfly

Example: can build 320Gbps router with 32 'current' servers

Implementation: the RB8/4



Specs.

- 8x 10Gbps external ports
- form-factor: 4U
- power: 1.2KW
- cost: ~$10k

Key results (realistic traffic)

- 72 Gbps routing
- reordering: 0-0.15%
- validated VLB bounds

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?