Big, Fast Routers

Dave Andersen
CMU CS 15-744

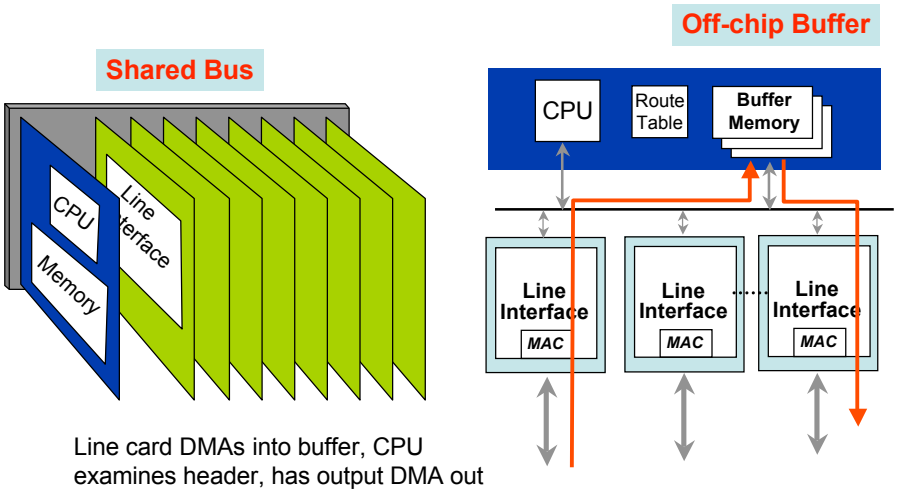
Router Architecture

- Data Plane
  - How packets get forwarded
- Control Plane
  - How routing protocols establish routes/etc.

Processing: Fast Path vs. Slow Path

- **Optimize for common case**
  - BBN router: 85 instructions for fast-path code
  - Fits entirely in L1 cache
- Non-common cases handled on slow path
  - Route cache misses
  - Errors (*e.g.,* ICMP time exceeded)
  - IP options
  - Fragmented packets
  - Mullticast packets

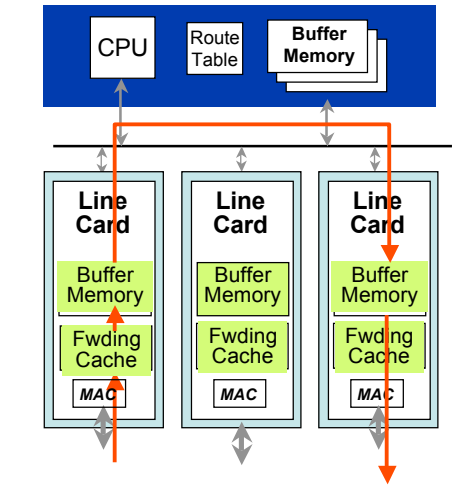
First Generation Routers



Line card DMAs into buffer, CPU examines header, has output DMA out

Typically <0.5Gb/s aggregate capacity

Second Generation Routers



Bypasses memory bus with direct transfer over bus between line cards

Moves forwarding decisions local to card to reduce CPU pain

Punt to CPU for "slow" operations

Typically <5Gb/s aggregate capacity

Control Plane & Data Plane

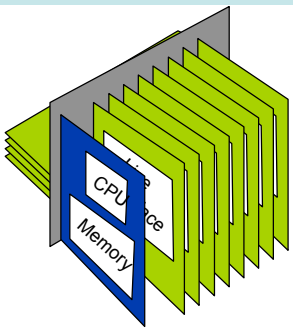
- Control plane must remember lots of routing info (BGP tables, etc.)
- Data plane only needs to know the "FIB" (Forwarding Information Base)
  - Smaller, less information, etc.
  - Simplifies line cards vs the network processor

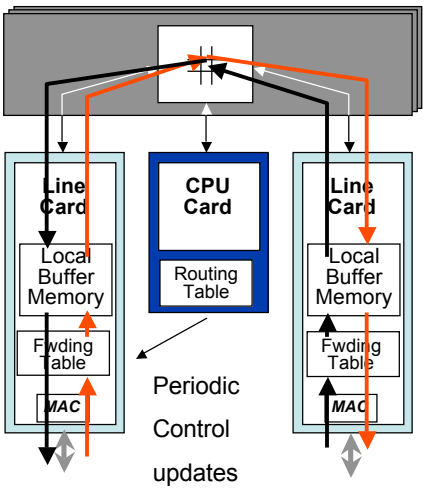
Bus-based

- Some improvements possible
  - Cache bits of forwarding table in line cards, send directly over bus to outbound line card
- But shared bus was big bottleneck
  - E.g., *modern* PCI bus (PCIx16) is only 32Gbit/sec (in theory)
  - Almost-modern cisco (XR 12416) is 320Gbit/sec.
  - Ow! How do we get there?

Third Generation Routers

"Crossbar": Switched Backplane





Typically <50Gb/s aggregate capacity

Crossbars

- N input ports, N output ports
  - (One per line card usually)
- Every line card has its own forwarding table/classifier/etc – removes CPU bottleneck
- Scheduler
  - Decides which input/output port to connect in a given *time slot*
  - Crossbar constraint:
    - If input I is connected to output j, no other input connected to j, no other output connected to input I
    - Scheduling is bipartite matching…

What's so hard here?

- Back-of-the-envelope numbers
  - Line cards can be 40 Gbit/sec today (OC-768)
    - Undoubtedly faster in a few more years, so scale these #s appropriately!
  - To handle minimum-sized packets (~40b)
    - 125 Mpps, or 8ns per packet
    - But note that this can be deeply pipelined, at the cost of buffering and complexity. Some lookup chips do this, though still with SRAM, not DRAM. Good lookup algos needed still.
- For every packet, you must:
  - Do a routing lookup (where to send it)
  - Schedule the crossbar
  - Maybe buffer, maybe QoS, maybe filtering by ACLs

Routing Lookups

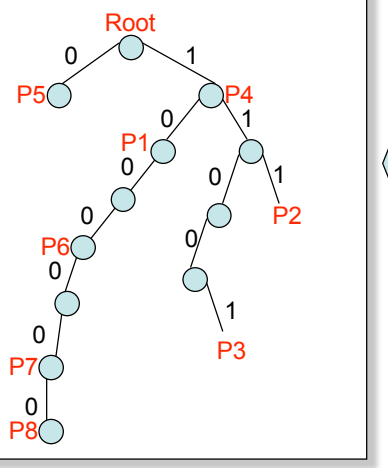
- Routing tables: 200,000 – 1M entries
  - Router must be able to handle routing table load 5 years hence. Maybe 10.
- So, how to do it?
  - DRAM (Dynamic RAM, ~50ns latency)
    - Cheap, slow
  - SRAM (Static RAM, <5ns latency)
    - Fast, $$
  - TCAM (Ternary Content Addressable Memory – parallel lookups in hardware)
    - Really fast, quite $$, lots of power

Longest-Prefix Match

- Not just one entry that matches a dst
  - 128.2.0.0/16 vs 128.2.153.0/24
  - Must take the “longest” (most specific) prefix

Method 1: Trie

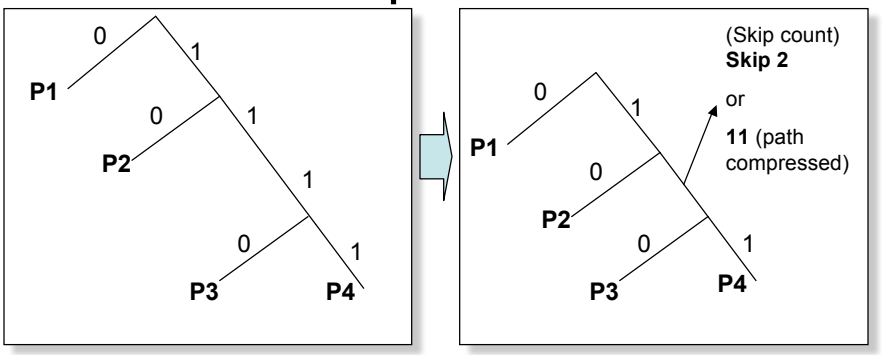
Trie



Sample Database

- P1 = 10*
- P2 = 111*
- P3 = 11001*
- P4 = 1*
- P5 = 0*
- P6 = 1000*
- P7 = 100000*

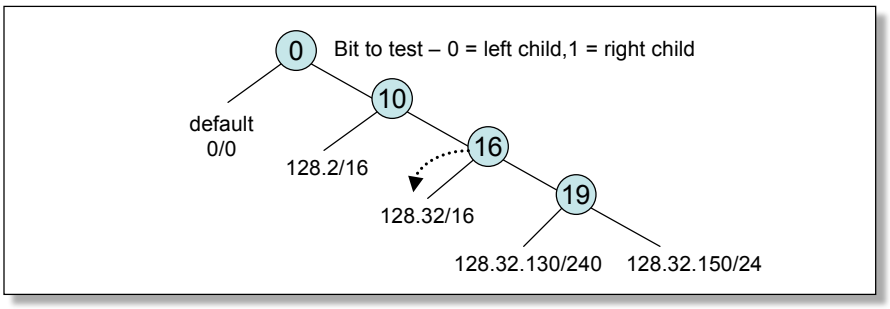
Skip Count vs. Path Compression



- Removing one way branches ensures # of trie nodes is at most twice # of prefixes
- Using a skip count requires exact match at end and backtracking on failure → path compression simpler

LPM with PATRICIA Tries

- Traditional method – Patricia Tree
  - Arrange route entries into a series of bit tests
- Worst case = 32 bit tests
  - Problem: memory speed, even w/SRAM!



How can we speed LPM up?

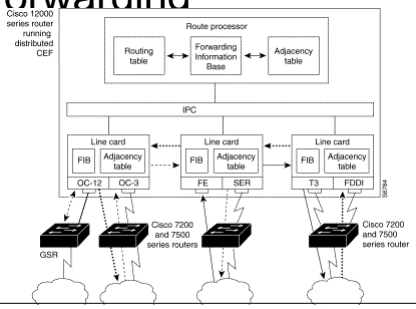
- Two general approaches:
  - Shrink the table so it fits in *really* fast memory (cache)
    - Degermark et al.; optional reading
    - Complete prefix tree (node has 2 or 0 kids) can be compressed well. 3 stages:
      - Match 16 bits; match next 8; match last 8
  - Drastically reduce the # of memory lookups
    - WUSTL algorithm ca. same time (Binary search on prefixes)

TCAMs for LPM

- Content addressable memory (CAM)
  - **Hardware-based** route lookup
  - Input = tag, output = value
  - Requires exact match with tag
    - Multiple cycles (1 per prefix) with single CAM
    - Multiple CAMs (1 per prefix) searched in parallel
  - Ternary CAM
    - (0,1,don't care) values in tag match
    - Priority (*i.e.*, longest prefix) by order of entries
  - Very expensive, lots of power, but fast! Some commercial routers use it

Skipping LPMs with caching

- Caching
  - Packet trains exhibit temporal locality
  - Many packets to same destination. Problems?
- Cisco Express Forwarding

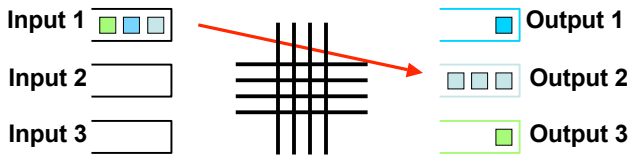


Problem 2: Crossbar Scheduling

- Find a bipartite matching
  - In under 8ns
- First issue: Head-of-line blocking with input queues
  - If only 1 queue per input
  - Max throughput <= (2-sqrt(2)) =~ 58%
- Solution? Virtual output queueing
  - In input line card, one queue per dst. Card
  - Requires N queues; more if QoS
  - The Way It's Done Now ™

Head-of-Line Blocking

Problem: The packet at the front of the queue experiences contention for the output queue, blocking all packets behind it.



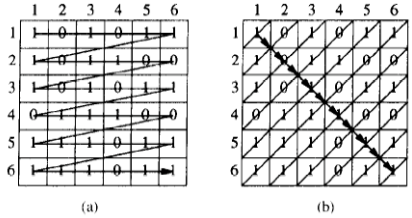
Maximum throughput in such a switch: 2 – sqrt(2)

Early Crossbar Scheduling Algorithm

- Wavefront algorithm

Slow!

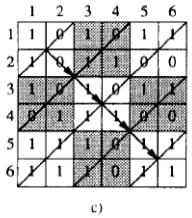
(36 “cycles”)



Observation:

2,1 1,2 don’t conflict with each other

(11 “cycles”)



Do in groups, with groups in parallel

(5 “cycles”)

Can find opt group size, etc.

Problems: Fairness, speed, …

Alternatives to the Wavefront Scheduler

- PIM: Parallel Iterative Matching
  - **Request:** Each input sends requests to all outputs for which it has packets
  - **Grant:** Output selects an input at random and grants
  - **Accept:** Input selects from its received grants
- **Problem:** Matching may not be maximal
- **Solution:** Run several times
- **Problem:** Matching may not be “fair”
- **Solution:** Grant/accept in round robin instead of

iSLIP – Round-robin PIM

- Each input maintains round-robin list of outputs
- Each output maints round-robin list of inputs
- Request phase: Inputs send to all desired output
- Grant phase: Output picks first input in round-robin sequence
  - Input picks first output from its RR seq
  - Output updates RR seq only if it's chosen
- Good fairness in simulation

100% Throughput?

- Why does it matter?
  - Guaranteed behavior regardless of load!
  - Same reason moved away from cache-based router architectures
- Cool result:
  - Dai & Prabhakar: *Any* maximal matching scheme in a crossbar with 2x speedup gets 100% throughput
  - Speedup: Run internal crossbar links at 2x the input & output link speeds

Filling in the (painful) details

- Routers do more than just LPM & crossbar
  - Packet classification (L3 and up!)
  - Counters & stats for measurement & debugging
  - IPSec and VPNs
  - QoS, packet shaping, policing
  - IPv6
  - Access control and filtering
  - IP multicast
  - AQM: RED, etc. (maybe)
  - Serious QoS: DiffServ (sometimes), IntServ (not)

Other challenges in Routing

- Fast Classification
  - src, dst, sport, dport, {other stuff} ->
    - accept, deny, which class/queue, etc.
  - Even with TCAMs, hard
    - efficient use of limited entries, etc.
- Routing correctness
  - We'll get back to this a bit later
- Architecture w.r.t. management
  - Also later. How do you manage a collection of 100s of routers?

Going Forward

- Today's highest-end: Multi-rack routers
  - Measured in Tb/sec
  - One scenario: Big optical switch connecting multiple electrical switches
    - Cool design: McKeown sigcomm 2003 paper
- BBN MGR: Normal CPU for forwarding
  - Modern routers: Several ASICs