6.896 Lecture 11: High-speed routing and forwarding

Hari Balakrishnan

October 14, 1998

1 Outline

- **Readings: DE+94 (for PIM), P+98, WVTP97.**
- CBT and PIM
- 50 Gb/s router from BBN.
- Fast lookups (overview).

2 CBT and PIM

- Problems with RPM schemes like DVMRP:
  1. Poor scaling: periodic flooding of packets. $O(S * G)$ state in routers. And routers at edge of tree also maintain lots of state.
  2. Does not handle "sparse membership" very well (sparse means widely distributed, does not necessarily mean that it's small scale).
- Core Based Trees (CBT)
  - Key idea: use one shared tree per group. Pick tree induced via subset of *core routers* in the network.
  - To join group, send packet to a well-known core router. Packet propagates towards core, and at some point joins existing tree. Now you're grafted on to the existing tree.
  - Notice that this is independent of the source.
  - However, using CBT in a "dense mode" situation will be sub-optimal.
- Protocol Independent Multicasting (PIM)

- Q: Why is it called PIM? (Independent of which unicast routing protocol is present. Else need DVMRP, MOSPF,...).
- Driven by need to provide efficient sparse mode delivery, as well as efficient dense mode delivery.
- PIM-SM: routers "meet" sources at well-known *rendezvous points (RP)*. Unless a router downstream sends an explicit join message up towards sparse mode distribution tree, no traffic is (ever) forwarded that way.
- Each SM router maintains some state. Q: what does it have to maintain? Each multicast group has a single RP.
- PIM-DM similar to DVMRP. Differences include: (1) independence from RIP or other unicast protocols, (2) forward-until-prune, rather than figure out children for whom you're on the shortest path back to the source.
- SM and DM groups use different portions of the IP multicast address space.

3 A 50 Gb/s router

3.1 Why bother?

- Trends
  1. Faster and faster link bandwidths.
  2. Larger and larger network size (hosts, routers, users).
  3. Users want more and more (15% increase in per-user demand in 1 year!).
  4. Not just because of Web: for example, Internet size has been growing at 50% per year since at least 1984!
  5. Conclusion: Unlike other areas in computer science that can rely on Moore's law to achieve progress, networking cannot!
- *Conventional dogma:* IP routers are inherently slow. They cannot possibly forward packets fast enough, where "fast" refers to multi-gigabit (or higher) rates.
- (As a corollary, this dogma concludes that we need pure ATM or equivalent in the backbone, and maybe end-to-end to achieve the dream of high-speed networks).
- This paper's motivation, at least in part, is to refute this widely held belief. If their results hold in actual deployment, then high-speed IP routers are far from dead!

3.2 How does it work?

- Routers do two things: participate in routing protocols with their neighbors, and forward packets. We are concerned primarily with the latter, since it's the data path.
- At first sight, forwarding is straightforward:
  1. Router gets packet.
  2. Looks at packet header for destination.
  3. Looks up routing table for output interface.
  4. Modifies header (ttl, IP checksum).
  5. Passes packet to output interface.
- But there's a lot of stuff to take care of: RFC 1812 is 175 pages long!
- **Challenge:** how to do all this *fast.* In particular, the common case.
- **Architecture:** (Draw picture).
  1. Multiple line cards.
  2. Multiple forwarding engines.
  3. High-speed switch fabric and processor.
  4. Network processor.
- Salient features of architecture:
  1. Separate forwarding engine from line card. From input line card, send only header to engine. Engine returns modified header to input line card, after which it gets vectored to output line card.
  2. Each forwarding engine has its own routing table (cache).
  3. Use a switched backplane rather than a shared bus. Custom-designed for IP rather than an ATM switch.
  4. QoS processing in router (on output line card).
  5. An abstract link-layer to handle different link technologies.
- **Forwarding engines:**
  - Key design principle: *optimize the common case.*
  - Not ASIC but based on Alpha 21164 at 415 MHz.
  - 8 KB I and D caches: forwarding code.

  - 96 KB L2 cache (Scache): cache of routing table. Only relevant table entries. 12000 routes (95% hit rate).
  - 16 MB L3 cache (Bcache): complete forwarding table in 2 8 MB chunks. Q: why is it banked like this?
  - Why not an ASIC or embedded processor? Want software. And chip has a very high clock speed and large Icache and Scache.
  - Optimize common case in custom assembler. 85 instructions to forward a packet.
  - Note: Does not check IP header checksum. Q: is this an important drawback?
  - Not-so-common cases deferred to network processor. This includes multicast.
  - Q: Do we need a route cache in the future? A: depends on what algorithm you choose to forward.

- **Switching fabric:** (Draw pic. of switch)
  - Advantage over bus? Parallelism.
  - Disadvantage? Multicast is hard. Need scheduling machinery. (They call this "allocation").
  - Input-queued switch. Q: what's the problem with standard input-queued switch? A: h-o-l blocking.
  - How to fix this? Keep track of per-output port traffic. Each input port bids for different outputs. In each epoch, match an input to output port.
  - Allocator arbitrates amongst bids to do this. Maximize switch throughput (rather than bounded latency).
  - Scheduling is pipelined (4 stages): get bids, allocate schedule, notify pairings, ship bits.
  - Q: What graph problem does this correspond to? A: bipartite matching.
  - Flow control by destination ports. Prevents an input from hogging an output.

- **Allocation algorithm:**
  - Heavily studied problem. Approaches include Parallel Iterative Matching (from DEC), wavefront scheduling, etc.
  - Use *randomization* to alleviate unfairness issues.
  - Essentially a greedy algorithm. Draw picture. Q: Is it optimal? A: nope, not at all!
  - Prioritize forwarding engine inputs over line cards (by skewing shuffling of ports).

- **Line cards:**
  - Input: simple, except for ATM and multicast.

  - Output: QoS processor for scheduling and queue management. A VLIW FPGA.

- **Network processor:**
  - Alpha processor running NetBSD.
  - Handles "not-so-common" cases of packet processing.
  - Handles routing protocols (gated).
  - Builds forwarding tables and populates table for each forwarding engine using a subset of each entry.
  - Using 2 banks in forwarding engines reduces disruptions and flapping effects.

3.3 Performance

- Peak performance of about 50 Gbps. This is under high utilization.
- Note: this is only an estimate, not deployed as yet.

3.4 Critique

- Sub-optimal switch allocation may end up affecting average case throughput.
- Actually, does not exclude (satisfactorily) forwarding engine to line card transfers in calculations of peak throughput estimates. I.e, the calculation is done by looking at spare capacity, rather than by looking at how the allocation would work. This is esp. important because of the higher priority given to forwarding engine input ports.
- Impressive design with persuasive arguments to motivate choices (mostly).

4 Fast lookups

- Two approaches from Sigcomm 97 papers
  1. Make better use of caching via compression (Degermark et al.).
  2. Constant-time lookups via clever data structures (Waldvogel et al.).
- Problem: Best-matching prefix.
- Basic idea: use hash tables for each prefix length. Then, apply binary search techniques to get to the longest matching prefix to find output interface.
- Lots of details to take care of. See paper.