Network
What’s in a router?

- Queuing delay
- Bottleneck queue (max size B)
- Link rate
- Packet-switched core network

Flows

What's in a router?
Routing Algorithm data plane control plane

4.1 • OVERVIEW OF NETWORK LAYER

309

In this example, a routing algorithm runs in each and every router and both forwarding and routing functions are contained within a router. As we'll see in Sections 5.3 and 5.4, the routing algorithm function in one router communicates with the routing algorithm function in other routers to compute the values for its forwarding table. How is this communication performed? By exchanging routing messages containing routing information according to a routing protocol! We'll cover routing algorithms and protocols in Sections 5.2 through 5.4.

The distinct and different purposes of the forwarding and routing functions can be further illustrated by considering the hypothetical (and unrealistic, but technically feasible) case of a network in which all forwarding tables are configured directly by human network operators physically present at the routers. In this case, no routing protocols would be required! Of course, the human operators would need to interact with each other to ensure that the forwarding tables were configured in such a way that packets reached their intended destinations. It's also likely that human configuration would be more error-prone and much slower to respond to changes in the network topology than a routing protocol. We're thus fortunate that all networks have both a forwarding and a routing function!
Hardware Router overview

Line Termination → Parsing → Lookup & Modification → [Match-Action] → Buffering & Scheduling → Line Termination

Input port → Switching fabric → Output port

Input port → ... → Output port

Input port → ... → Output port
Study 2 designs

• Historically evolving, multiple concurrent router designs

• 2 exemplars:
  • MGR: router from the late 1990s (50Gbit/s router, Partridge et al)
  • RMT: router from the late 2010s
Life of a Packet
(2) Packet parsing

• Dividing a sentence into its grammatical parts:
  • “I ate an apple”
  • Sentence := Subject (I) Verb (ate) Object (an apple)
  • Object:= Article (an) Noun (apple)

• Packet parsing: dividing sequence of bits into header fields
  • \[1100100101011\]
    • \(\Rightarrow\) ethernet destination (1100) l source (1001) l ethertype (01) l …
  • Unlike parsing English, parsing packets is quite mechanical
  • Series of extractions and branches to assign fields to packet bits

• Parsing could be implemented in software (MGR) or hardware (RMT)
(2) Packet parsing

• A key principle: **Separate header and payload data paths**
  • Router functionality is “header-heavy” but “payload-light”
  • Don’t move the payload around too much
  • Conserve bandwidth & resources for data moved inside the router

• Header goes on as input to route lookup/packet modification

• Payload sits on a buffer until router knows what to do with pkt
  • Buffer could be on the ingress line card (MGR) or a buffer shared between line cards (RMT)
Parsing state machine

- Parsing is an inherently **sequential** process
- Previous header determines the next header type
- Current header length determines the start of the next header
- Parser state: tracks the current header and its length
  - Help jump to the next state

Source: Design principles for packet parsers, Gibb et al.
A hardware (fixed function) parser

• Two steps:
  
  • **Header identification:** Identify sequence of headers
  
  • **Extract fields** from identified headers to send to matching components of tables
  
  • Design digital circuits with a high clock rate?
(Fixed function) header identification

- Identify headers through fixed-function header processors
- Simple design: extract one header per clock cycle
- **Speculate** to extract multiple headers/cycle
- Sequence resolution picks one
(Fixed function) field extraction

- Extract fields using fixed offsets into the packet, depending on parser state

- Field-extraction table is hard-coded for specific fields and protocols
  - E.g., IPv4 length: always at bytes 3 & 4
RMT makes parsing programmable

• Parse graph: Representation of all valid header sequences you expect
Programmable parser

- Reprogrammable TCAM allows us to identify headers based on the protocol parse graph

- Reprogrammable SRAM contains the protocol-dependent field extractions:
  - e.g., IPv4, byte 8 → TTL, 12—15 → IPv4 source address, etc.
  - Extracting the next header & length is a special case of generic field extractions for each protocol
Output of programmable parser

• Vector of extracted packet fields and labels. Example:
  • Ethernet -> src: 1010..; dst: 0101…; ethertype: IPv4
  • IPv4 -> version: …; …
  • … <other headers>

• Termed the packet header vector (PHV)

• Fed into the packet lookup and modification engines
(3) Packet Lookup & Modification

- The main job of a router is to forward packets to the correct output port(s)
- Typically done by looking up a table of entries that were pre-computed by the control plane
- Look-up tables with a range of sizes (# entries), widths (headers)
- Packets may also need to be modified
  - RFC 1812 TTL decrement, recompute IP checksum, MAC rewrite, …
  - Virtualized public cloud: encaps/decaps headers
- Outcome: a (set of) output ports + (possibly modified) headers
Typical table structures

• Sequence of tables: L2, L3, ACL
• Ethernet (L2) headers to forward packets within one IP network
• IP (L3) headers to forward packets across IP networks
• Access control lists (ACL) to implement firewalls & other policies
• Different kinds of lookups possible:
  • Exact match
  • Longest prefix match
  • Wildcard match
Packet lookup in MGR

- A forwarding engine card separate from line cards
  - Scale forwarding and interface capacity separately
- **Software:** Use Alpha 21164 (a 415MHz generic processor)
- Programmed in assembly to do route lookup and other processing.
- Many optimizations to improve performance
  - Need for such optimizations continues today for software
MGR Software lookup performance

- Separate fast path from slow path (optimize the common case)
  - ARP lookup, fragmentation, error handling
- Try to fit all code into the processor instruction cache
- Heavily use caching for table entries across different memories in a hierarchy
  - Traffic locality: a small fast memory can service “most” traffic
- Two copies of table in external memory to support downtime-less updates to the forwarding table
- However, can’t guarantee deterministic throughput for packets
  - Packet might access slower memories in the memory hierarchy
Packet lookup in RMT: *Pipelined parallelism*

- Different functionalities (ex: L2, L3) in different table stages
- Highly parallel over packets: admit 1 packet/cycle
- Pipeline circuitry *clocked* at a high rate: ex: RMT@1 GHz
- Deterministic throughput
Traditional pipelined hardware: fixed-function (Multiple Match-Action Tables)
MMT isn’t enough!

• Operators want new protocols and services

• **Virtualization** and the cloud
  • VMs have their own address space (e.g., 10.0.x.x)
  • Physical networks route traffic using a different set of addresses (e.g., 128.6.x.x)
  • Keep them separate so you can place VMs wherever you can
  • Need: **dedicated new packet headers** to use for forwarding within the “core” fabric of the network
    • E.g., VXLAN, NVGRE

• Research experimentation and domain-specific headers
  • E.g., finance; university campuses; network feedback signals
MMT isn’t enough!

• Want to use table memory flexibly
  • Different environments need tables of different shapes and sizes
• Enterprises: ACL-heavy (“students can see information sent to other students from professors, but cannot see info from professors to printers”)
• Tier-1 ISP like ATT: L3-heavy (~1 million Internet IPv4 prefixes)
• Static table sizes don’t work well
  • Can’t use another table’s memory, even if it is empty
  • Heterogenous devices to support different scenarios: complexity
  • Even device for a specific market may have insufficient resources
MMT isn’t enough!

• All of this might be supported if switch hardware can be upgraded as often as software in general (e.g., on smartphone)

• Unfortunately, the reality is very far from it:
  • Each device generation hardware upgrade: 3--5 years
    • New ASIC design, verification, fabrication, testing
  • Even software upgrades take time:
    • Features requested by other customers stand in the way of releasing new feature that one customer wants
RMT: Protocol Independent Switch Arch

Stages of Match-Action Tables

Programmable Parser (Gibb et al, ANCS’13)

Programmable Match-Action Pipeline (RMT, SIGCOMM’13)

Programmable scheduler (PIFO, SIGCOMM’16)

To egress pipeline
A primer on high-speed memories

• Computers have different kinds of memory
  • Fastest caches (L1, L2, ...) made of SRAM
  • Fast main memory made of DRAM
  • Storage (HDD) made of magnetic disks, tape, SSDs, and so on
  • Translation Lookaside Buffer (TLB) with CAM/TCAM

• Today’s high-speed routers use **SRAM** and **TCAM**
  • Static Random Access Memory for exact match
  • Ternary Content Addressable Memory for wildcard ACL match (also longest prefix match)
SRAM principles of operation

- Memory is organized into **banks**
- Each bank can be independently accessed through an **address**
- Data in the memory row at the address can be read/written each clock cycle
- Banks of larger sizes are denser (fewer wires to run), but you can only read/write one data row per bank per clock cycle (reduced parallelism)
- **Looking up** (ex: IPv4 dst) involves computing address(es) & reading mem
TCAM principles of operation

- Banks are accessed using content, not addresses (CAM)
- Contents of the memory are ternary: values can be 0, 1, or x (don’t care)
- Incoming keys are matched against TCAM, with any bit accepted at the location of the wildcard bits
- Ternary logic is power- and area-hungry relative to SRAM