Network
Life of a packet in a hardware router

1. **Line Termination**
2. **Parsing**
3. **Lookup & Modification**
   - (Match-Action)
4. **Buffering & Scheduling**
5. **Line Termination**

Diagram:
- **Input port**
  - ... (Multiple)
- **Output port**
  - ... (Multiple)
- **Switching fabric**
Pipelined Parallelism

(a) Parsing a new packet.

(b) The Ethernet next-header field identifies the IPv4 header.

(c) The IPv4 length field identifies the IPv4 header length.

(d) The IPv4 next-header field identifies the TCP header.

“SRAM port”

address

data row
RMT: Match-Action Table memory design

- Match and Action units supplied with the Packet Header Vector
- Each stage accesses its own memory containing tables

- RMT uses a crossbar to pick PHV fields for matching against contents of SRAM/TCAM banks
  - Flexible key generation for lookup

- Distinction from fixed-function:
  - User-programmed fields stored in table
  - PHVs include user-programmed fields
  - Table match keys chosen by users
RMT: Match-Action Table memory design

- Entry looked up in the memory (SRAM/TCAM) contains a pointer to the instruction (action) for that entry

- Instructions implemented through programmable ALUs
  - More general ALUs than fixed-function hardware devices
- Feasible since compute components “cheap” inside a router
- VLIW: operate on multiple headers simultaneously

- Entry also contains pointers to:
  - Action data (e.g., output port),
  - statistics (counters), if needed

PHV → ALUs
Achieving reconfigurable match-action

Separately configurable and addressable memory blocks is the key to using tables flexibly. Independent block level access.

Fixed-function matching

Flexible match function

Each table in a match-action stage may use a different crossbar configuration to extract a different set of fields from the PHV.

Different memory banks can contain entries for different tables (e.g., IPv4 matching, virtualization, …)

Cost: 14% extra area (& power) for fatter wires.
Memory layout and use matters

- Flexible partitioning of memory across SRAM and TCAM
- Fixed size memory blocks: internal fragmentation
- Deterministic access times
  - All of it is SRAM or TCAM
- Interesting compiler concerns
  - “Packing” tables
  - Within & across pipeline stage
Routing Algorithm

data plane
control plane

4.1

Routing algorithms determine values in forwarding tables. 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!

Who programs the rules? Control plane

Control plane
per route-change processing (~ a few seconds)

Data plane
per-packet processing (~ tens of nanoseconds)
Distributed control planes
Software-defined network (SDN)

Logically-centralized control plane

Data plane

Data plane
SDN (1/2): Centralized control plane

SDN controller

Control planes lifted from switches into a logically centralized controller
SDN (2/2): Open interface to data plane
Packet after route lookup

• What we have after lookup and modifications:
  • One or more output ports, or a decision to drop
  • packet headers, possibly modified from ingress

• Goal: reconstitute the headers with payload and send the packet out of one or more output ports

• Move the packet (header) to output through the switching fabric
Building a high-speed switching fabric
A single (n X m)-port switching fabric

• Different designs of switching fabric possible
• Assume n ingress ports and m egress ports, half duplex links
A single (n X m)-port switching fabric

• We want a design such that:

• Any port can connect to any other directly if all other ports free

• Nonblocking: if input port x and output port y are both free, they should be able to connect
  • Regardless of other ports being connected.
  • If not satisfied, switch is blocking.
Nonblocking designs are nontrivial

Two aspects: **topology** and **routing**
High port density + nonblocking == hard!

- Low-cost nonblocking crossbars are feasible for small # ports

- However, it is costly to be nonblocking with many ports

- If each crossover is as fast as each input port,
  - Number of crossover points == n * m
  - Cost grows quadratically on the number of input ports

- Else, crossover must transition faster than the port
  - … so that you can keep the number of crossovers small
Nonblocking switches with many ports

• Key principle: Build a fast, nonblocking switch with many ports using many fast, nonblocking switches with a small number of ports.

• How to build large nonblocking switches?
  • The subject of interconnection networks
  • https://en.wikipedia.org/wiki/Multistage_interconnection_networks
3-stage Clos network (r*n X r*n ports)
How Clos networks become nonblocking

• Adjacent layers of Clos are fully connected with each other

• if \( m > 2n - 2 \), then the Clos network is strict-sense nonblocking
  • Cost: you need more output ports than input ports

• That is, any new demand between any pair of free (input, output) ports can be satisfied without re-routing any of the existing demands.
Need at most \((n-1)+(n-1)\) middle stage

At most \(n-1\) existing demands
Surprising result about Clos networks

• if $m \geq n$, then the Clos network is rearrangeably nonblocking

• Any new demand between a pair of free (input, output) ports can be satisfied by suitably re-arranging existing demands.
  • Re-arranging == re-routing

• It is easy to see that $m \geq n$ is necessary
  • The surprising part is that $m \geq n$ is sufficient
  • Intuition: matching in a fully connected bipartite graph
Rearrangeably nonblocking Clos built with identical switches: $n^2 \times n^2$ using $3n \times n$
What about re-routing?

• How to rearrange existing demands when a new packet arrives, so that it can get across as quickly as possible?

• Ideally, do it without “interference” to (i.e.: rerouting) other pkts
  • In general, it is not possible. Packets must wait, adding to delays.

• But, doable with high probability based on the input traffic workload: if packets are “randomly shuffled” across output ports
  • Valiant Load Balancing
Valiant Load Balancing (VLB)

• Suppose you had many paths to get to a destination
• Key idea: pick a random node to redirect a message to from the source, then follow a deterministically-chosen shortest path to the destination from there
• Guarantee: With high probability, the message reaches its destination “very quickly” without blocking (log n steps in a n-hypercube)
• Practically, this means close to zero queueing delays in switch
• Randomized algorithms can be powerful.
VLB and Clos

- VLB is more general than Clos
  - It is a form of *oblivious routing*
  - e.g., no need to measure traffic patterns before choosing routes
  - Extremely simple to implement: no global state
- VLB is handy in Clos topologies due to the *numerous options to pick the intermediate layer switch*
  - Balance load across many paths
- Very beneficial in practice
  - “Performance isolation:” other port – port flows don’t matter
  - High capacity (“bisection bandwidth”) between two ports