We have described the "end host" aspects of Routing (IP) and congestion control (TCP). But these need support from the network, i.e., the Routers.

Roughly speaking, router architecture (logically):

(I) Data path of packets: packet arrives at a "line card" or input "port"; its next-hop is looked up using Routing Table information [remember the routing algorithm] and then placed in appropriate VOA at the input port.

(II) Switch, transfer packets from inputs to output, where packets may be queued.

(III) The queued packets on outgoing line are served as per some "fair share".
The algorithms for (I), (II) and (III) primarily constitute the "intelligence" of Routers.

Today, we shall describe algorithm for (I). The next lecture will be for algorithm (III). Recall that algorithms for (II), scheduling, was discussed as part of the physical layer algorithms.

Algorithm for (I): Route Lookup and Packet Classification

- Route lookup: determine the next hop of an arriving packet based on the local routing table.

- Packet classification: classify packet into various categories based on its attributes like source IP, etc. Simplest form of the classification is "FireWall" which "black list" which packets are not to be admitted.

Today, we shall learn about these two algorithms.

*Route lookup*: Logically, each packet has its destination IP address as part of its header. The IP address consists of 32 bits, each in the "block" of 8 bit, 4 blocks. E.g., 128.32.17.1, or

```
10000000 00100000 00010000 00000000
```

Now Routing table in principle may require to store all
Only few outgoing ports. Therefore, really speaking each router need to maintain the next hop information only into "clusters" of addresses. For example, a possible rule could be = all dest addresses with 128.x.x.x should be forwarded to port 6.

This has motivated, what is called, Longest Prefix Matching, rule.

Basic Insight: the later bits are differentiating only *local* routing information and the longest match with next hop rule provides most specific routing information.

Thus, the routing table stores information as follows:

<table>
<thead>
<tr>
<th>Rules</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 128. <em>.</em>.*</td>
<td>Port 1</td>
</tr>
<tr>
<td>R2 128.4.<em>.</em></td>
<td>Port 2</td>
</tr>
<tr>
<td>R3 126.2.<em>.</em></td>
<td>Port 1</td>
</tr>
</tbody>
</table>

Thus, if packet with 128.3.2.1 as dest is matched to R1 and hence routed to port 1. But packet with 128.4.3.1 is matched with R1, R2. Since R2 is more specific (longer prefix match) it will be sent to port 2.

Finally, packet with dest 126.4.3.1 with match in 3 parts to R1, R2, R3. But it has prefix match with only R3 and is the most specific so it will dest to port 1.
Main algorithmic challenge: extremely quick efficient lookup of packets' next hop. The question is what type of data structure and how to lookup as well as how to keep it updated.

There are many interesting solutions. A reasonable survey of architectures is available in Thesis by Dr. Pankaj Gupta, Stanford University.

However, the 'prime' solution is based on a hardware construct called Ternary Content Addressable Memory (TCAM) because \((0, 1, x)\).

TCAM: it is an array of memory that in a sense does parallel "search". Operationally, it works as follows:

```
Address | next hop
------- |--------
A1      | H1     
A2      | H2     
...     | ...    
An      | Hn     
```

Design: TCAM, upon query with an address \(A\) outputs all the next hop info of the rows/rules that have prefix match. But we want the longest, prefix that matched.
A simple solution: Organize all addresses in the order of their length of the prefixes.

Update: the IP address lookup rules or Routing Tables are updated every few hours. This leads to either (a) down time for rule updates or (b) very expensive system design.

We would like an algorithm that can incrementally keep the TCAM updated.

Now if there are 64K entries in a TCAM, then update to maintain exact prefix ordering could take roughly 64K operations. But this may take time on order of 100ms. And eventually to facilitate ~40K (per 2ms) update, it may take time forever.

Observation: the conflicting rules must have different "length" of prefixes. For example, 128 addresses rules.

\[
\begin{align*}
128 \cdot * \cdot * & \quad \text{do not conflict.} \\
126 \cdot * \cdot * & \\
\end{align*}
\]

But

\[
\begin{align*}
128 \cdot 2 - \cdot * & \quad \text{do conflict.} \\
128 \cdot * \cdot * & \\
\end{align*}
\]

Maintaining EXACT conflict order is too much.
But, here is an *easy* solution.

Maintain all adder rules together as per their prefix lengths. Rules with different prefix lengths are separated in "pools".

For example:

\[
\begin{array}{c}
32\text{-bit prefix} \\
31\text{-bit prefix} \\
\vdots \\
1\text{-bit prefix} \\
\text{empty space}
\end{array}
\]

When update happens, say a rule with prefix 4 need to be inserted do:

\[
\begin{array}{c}
32\text{-bit prefix} \\
\vdots \\
4\text{-bit prefix} \\
3\text{-bit prefix} \\
2\text{-bit prefix} \\
1\text{-bit prefix} \\
\text{empty space}
\end{array}
\]

They total update (add/delete) op: are at most 32!

Next topic: packet classification.

We shall discuss simplest form of packet classification:

Find if given packet is allowed to enter or not. That is, “firewalling.”

We shall utilize a nice data structure called Bloom Filters. Nice survey by M. Mitzenmacher is available.

Basic construct:

We have total of m-bits space available.
We have to store n "rules" or items: x_1, ..., x_n.
In our context, this corresponds to attributed that are "forbidden."

Mechanism: there are k - random hash functions, f_1, ..., f_k:
They map each item x_i to k locations in m bit space:

f_1(x_i), ..., f_k(x_i): i = 1, ..., n.

Make all of these positions: (f_1(x_i), ..., f_k(x_i))_{i=1}^n
When a new item (say packet) \( q \) arrives, do:

- Check if bit positions \( f_1(q), \ldots, f_k(q) \) is 1.
  - If yes, term \( q \) is forbidden.
  - If no, let \( q \) enter.

**Invariant:** All forbidden \( q \) will not be entered.

**Flip side:** Some additional \( q \) will not be entered too.

**Advantage:** Quick lookup and small memory.

**Perfor:** \( q \) is not forbidden but it will not be admitted.

\[
\text{Perfor} = \mathbb{P}(q \text{ is not admitted}) \\
= \mathbb{P}(f_1(q) = 1, \ldots, f_k(q) = 1) \\
= \mathbb{P}(f_1(q) = 1)^k \\
= (1 - \mathbb{P}(f_1(q) = 0))^k \\
= \left(1 - \left(1 - \left(1 - \frac{1}{m}\right)^k\right)^k\right) \\approx (1 - e^{-k/m})^k
\]