Last Lecture: Network Layer

1. Design goals and issues
2. Basic Routing Algorithms & Protocols
3. Addressing, Fragmentation and reassembly
4. Internet Routing Protocols and Inter-networking
5. Router design
   1. Short History + Router Architectures
   2. Switching fabrics ✔
   3. Address lookup problem
6. Congestion Control, Quality of Service
7. More on the Internet’s Network Layer
Last Lecture: Network Layer

1. Design goals and issues
2. Basic Routing Algorithms & Protocols
3. Addressing, Fragmentation and reassembly
4. Internet Routing Protocols and Inter-networking
5. Router design
   1. Short History + Router Architectures
   2. Switching fabrics
   3. Address lookup problem ✔
6. Congestion Control, Quality of Service
7. More on the Internet’s Network Layer
Reminder on Router Architecture

Header Processing

- Lookup IP Address
- Update Header
- Queue Packet

- Address Table
  - IP Address
  - Next Hop

- Buffer Memory
  - ~1M packets
  - Off-chip DRAM

- Data
  - Hdr

- Data
  - Hdr

~1M prefixes
- Off-chip DRAM

~1M packets
- Off-chip DRAM
Basic Requirements

1) Fast lookup **and** fast update

2) Scalable (speed & table size)

3) Inexpensive (fit in memory, e.g.)
Lookups Must Be Fast

1. Lookup mechanism must be simple and easy to implement
2. Memory access time is the bottleneck
   \[ 200 \text{Mpps} \times 2 \text{ lookups/pkt} = 400 \text{ Mlookups/sec} \Rightarrow 2.5 \text{ns per lookup} \]

<table>
<thead>
<tr>
<th>Year</th>
<th>Aggregate Line-rate</th>
<th>Arriving rate of 40B POS packets (Million pkts/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>622 Mb/s</td>
<td>1.56</td>
</tr>
<tr>
<td>1999</td>
<td>2.5 Gb/s</td>
<td>6.25</td>
</tr>
<tr>
<td>2001</td>
<td>10 Gb/s</td>
<td>25</td>
</tr>
<tr>
<td>2003</td>
<td>40 Gb/s</td>
<td>100</td>
</tr>
<tr>
<td>2006</td>
<td>80 Gb/s</td>
<td>200</td>
</tr>
</tbody>
</table>
## Memory Technologies (2006)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Max single chip density</th>
<th>$/chip ($/MByte)</th>
<th>Access speed</th>
<th>Watts/chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking DRAM</td>
<td>64 MB</td>
<td>$30-$50 ($0.50-$0.75)</td>
<td>40-80ns</td>
<td>0.5-2W</td>
</tr>
<tr>
<td>SRAM</td>
<td>8 MB</td>
<td>$50-$60 ($5-$8)</td>
<td>3-4ns</td>
<td>2-3W</td>
</tr>
<tr>
<td>TCAM</td>
<td>2 MB</td>
<td>$200-$250 ($100-$125)</td>
<td>4-8ns</td>
<td>15-30W</td>
</tr>
</tbody>
</table>

Note: rough estimates only. Manufacturer & technology & market dependent.
(Ternary) Content Addressable Memory

search lines

matchlines

encoder

match address

matchline sense amps

search line drivers

search data = 0 1 1 0 1
## Lookup Problem: Protocol Dependent

<table>
<thead>
<tr>
<th>Networking Protocol</th>
<th>Lookup Mechanism</th>
<th>Techniques we will study</th>
</tr>
</thead>
</table>
| MPLS, ATM (virtual circuits) | **1. Exact match search** | – Direct lookup  
– Associative lookup  
– Hashing  
– Binary/Multi-way Search Trie/Tree |
| Ethernet |  |  |
| IPv4, IPv6 (datagram) | **2. Longest-prefix match search** | -Radix trie and variants  
-Compressed trie  
-Binary search on prefix intervals |
### 1. Exact Match – Virtual Circuit Reminder

**Forwarding table in northwest router:**

<table>
<thead>
<tr>
<th>Incoming interface</th>
<th>Incoming VC #</th>
<th>Outgoing interface</th>
<th>Outgoing VC #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>97</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Exact Matches in ATM/MPLS

- VCI/Label space is 24 bits
  - Maximum 16M addresses.
  - With 64b data, this is 1Gb of memory.
- VCI/Label space is private to one link
- Therefore, table size can be “negotiated”
- Alternately, use a level of indirection
Exact Matches in Ethernet Switches

- Layer-2 addresses are usually 48-bits long,

- The address is global, not just local to the link,
  - The range/size of the address is not “negotiable” (like it is with ATM/MPLS)

- \( 2^{48} \approx 262,144 \) Gig, *too large*, cannot hold all addresses in table and use direct lookup.
Ethernet Switches: Associative Lookups

- Associative memory (aka Content Addressable Memory, CAM) compares all entries in parallel against incoming data in one clock-cycle.
• Use a pseudo-random hash function (relatively insensitive to actual function)
• Bucket linearly searched (or could be binary search, etc.)
• Leads to unpredictable number of memory references
Performance of Lookup With Hashing

Expected number of memory references:

\[ ER = \frac{1}{2} \left( \frac{1}{1 + \alpha} \right) \]

\[ = \frac{1}{2} \left( \frac{1}{1 + \frac{\alpha}{1 - \left(1 - \frac{1}{N}\right)^M}} \right) \]
There always exists a “perfect” hash function (since # of hosts connected to the switch is $\leq 2^{16}$)

With a perfect hash function, memory lookup always takes O(1) memory references.

**Problem:**
- Finding perfect hash functions (particularly a minimal perfect hash) is complex.
- Updates make such a hash function yet more complex
- Advanced techniques: multiple hash functions, *bloom filters*...
Hashing: Pretty Good Choice for Exact Match

- **Advantages:**
  - Simple to implement
  - Expected lookup time is small
  - Updates are fast (except with perfect hash functions)

- **Disadvantages**
  - Relatively inefficient use of memory
  - Non-deterministic lookup time (in rare cases)

⇒ Attractive for software-based switches. However, hardware platforms are moving to other techniques (but they can do well with a more sophisticated form of hashing)
Trees and Tries

Binary Search Tree

- Lookup time dependent on table size, but independent of address length, storage is $O(N)$, assuming comparisons can be done in $O(1)$-time

Binary Search Trie

- Lookup time bounded and independent of table size, storage is $O(NW)$
Multiway Tries

16-ary Search Trie

Question: Why don’t we just make it a $2^{48}$-ary trie?

Ptr=0 means no children
Multiway Tries

As degree increases, more and more pointers are “0”

<table>
<thead>
<tr>
<th>Degree of Tree</th>
<th># Mem References</th>
<th># Nodes ($\times 10^6$)</th>
<th>Total Memory (Mbytes)</th>
<th>Fraction Wasted (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>48</td>
<td>1.09</td>
<td>4.3</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>0.53</td>
<td>4.3</td>
<td>73</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>0.35</td>
<td>5.6</td>
<td>86</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>0.25</td>
<td>8.3</td>
<td>93</td>
</tr>
<tr>
<td>64</td>
<td>8</td>
<td>0.17</td>
<td>21</td>
<td>98</td>
</tr>
<tr>
<td>256</td>
<td>6</td>
<td>0.12</td>
<td>64</td>
<td>99.5</td>
</tr>
</tbody>
</table>

Table produced from $2^{15}$ randomly generated 48-bit addresses
More on Tries

- **Advantages:**
  - Bounded lookup time
  - Simple to implement and update

- **Disadvantages:**
  - Inefficient use of memory and/or requires large number of memory references

  More sophisticated algorithms compress ‘sparse’ nodes.
2. Longest Prefix Match -- Reminder

ISPs-R-Us has a more specific route to Organization 1

Organization 0
200.23.16.0/23

Organization 2
200.23.20.0/23

Organization 7
200.23.30.0/23

Organization 1
200.23.18.0/23

ISPs-R-Us

"Send me anything with addresses beginning 200.23.16.0/20"

"Send me anything with addresses beginning 199.31.0.0/16 or 200.23.18.0/23"

Requires routers to do longest prefix match, per packet, every few nanosecond
## Example

<table>
<thead>
<tr>
<th>address</th>
<th>mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001111 01011100 00000000 10000111</td>
<td>11111111 11111111 11111111 11111111</td>
</tr>
<tr>
<td>11001111 01011100 00000000 00000000</td>
<td>11111111 11111111 00000000 00000000</td>
</tr>
<tr>
<td>11001111 01011100 00000000 00000000</td>
<td>11111111 11111111 11100000 00000000</td>
</tr>
</tbody>
</table>

**Packet’s Destination Address:**

11001111 01011100 00000000 10010111
Longest Prefix Matching (LPM) Is Hard!

- Arriving packet does not carry “network prefix”
  - Why?

- Hence, one needs to search among the space of all prefix lengths; as well as the space of all prefixes of a given length

- There are many proposed solutions, we’ll only talk about a few
Use ... 32 Exact Matching Algorithms

- Exact match against prefixes of length 1
- Exact match against prefixes of length 2
- Exact match against prefixes of length 32

Network Address

Priority Encode and pick

Port

Encode and pick
Objectives

- Speed (= number of memory accesses)
- Storage requirements (= amount of memory)
- Low update time (support >10K updates/s)
- Scalability
  - With length of prefix: IPv4 unicast (32b), Ethernet (48b), IPv4 multicast (64b), IPv6 unicast (128b)
  - With size of routing table: (sweetspot for today’s designs = 1 million)
- Flexibility in implementation
- Low preprocessing time
Radix Trie (Recap)

### Trie node

<table>
<thead>
<tr>
<th>next-hop-ptr (if prefix)</th>
<th>left-ptr</th>
<th>right-ptr</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Index</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>111*</td>
<td>H1</td>
</tr>
<tr>
<td>P2</td>
<td>10*</td>
<td>H2</td>
</tr>
<tr>
<td>P3</td>
<td>1010*</td>
<td>H3</td>
</tr>
<tr>
<td>P4</td>
<td>10101</td>
<td>H4</td>
</tr>
</tbody>
</table>

Lookup 10111

Add P5 = 1110*
Pros and Cons

- **W-bit prefixes**: $O(W)$ lookup, $O(NW)$ storage and $O(W)$ update complexity

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ Simplicity</td>
<td>➢ Worst case lookup slow</td>
</tr>
<tr>
<td>➢ Extensible to wider fields</td>
<td>➢ Wastage of storage space in <em>chains</em></td>
</tr>
</tbody>
</table>
Patricia Tries: Compress the Un-branched

Path-compressed tree node structure

<table>
<thead>
<tr>
<th>P1</th>
<th>111*</th>
<th>H1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>10*</td>
<td>H2</td>
</tr>
<tr>
<td>P3</td>
<td>1010*</td>
<td>H3</td>
</tr>
<tr>
<td>P4</td>
<td>10101</td>
<td>H4</td>
</tr>
</tbody>
</table>

variable-length bitstring | next-hop (if prefix present) | bit-position
---|---|---
left-ptr | right-ptr |

PATRICIA: Practical Algorithm To Retrieve Information Coded as Alphanumeric

SUNY at Buffalo; CSE 489/589 – Modern Networking Concepts; Fall 2010; Instructor: Hung Q. Ngo
Binary Search on Prefix Intervals [Lampson98]

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>/0</td>
</tr>
<tr>
<td>P2</td>
<td>00/2</td>
</tr>
<tr>
<td>P3</td>
<td>1/1</td>
</tr>
<tr>
<td>P4</td>
<td>1101/4</td>
</tr>
<tr>
<td>P5</td>
<td>001/3</td>
</tr>
</tbody>
</table>

SUNY at Buffalo; CSE 489/589 – Modern Networking Concepts; Fall 2010; Instructor: Hung Q. Ngo
Alphabetic Tree

0111

0011

0001

I

1

2

3

4

5

6

≤

> 1/8

≤

> 1/32

I

3

4

5

6

≤

> 1/2

1/4

≤

> 1/32

I

1

2

3

4

5

6

≤

> 1/16

1/32

P1

P2

P3

P4

P5

0000 0010 0100 0110 1000 1010 1100 1110 1111
Another Alphabetic Tree

$I_1$

1/2

$I_2$

1/4

$I_3$

1/8

$I_4$

1/16

$I_5$

1/32

$I_6$

1/32
Advantages

- Storage is linear
- Can be 'balanced'
- Lookup time independent of $W$

Disadvantages

- But, lookup time is dependent on $N$
- Incremental updates more complex than tries
- Each node is big in size: requires higher memory bandwidth

• W-bit N prefixes: $O(\log N)$ lookup, $O(N)$ storage
### Rough Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Build</th>
<th>Search</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(pure) Binary Search</td>
<td>(O(WN \log N))</td>
<td>(O(W \log N))</td>
<td>(O(NW))</td>
</tr>
<tr>
<td>Trie</td>
<td>(O(NW))</td>
<td>(O(W))</td>
<td>(O(NW))</td>
</tr>
<tr>
<td>Binary prefix search</td>
<td>(O(N \log W))</td>
<td>(O(\log W))</td>
<td>(O(N \log W))</td>
</tr>
</tbody>
</table>

But, those are not the only practical concerns. Update time, e.g., is important!