Efficient Addressing and Forwarding

Outline

Addressing
Subnetting
Supernetting
CIDR
Longest Prefix Match
Global Addresses

• Properties
  – IPv4 uses 32 bit address space
  – globally unique
  – hierarchical: network + host

• Dot Notation
  – 10.3.2.4
  – 128.96.33.81
  – 192.12.69.77

• Assigning authority
  – Jon Postel ran IANA ‘til ‘98
  – Assigned by ICANN
How to Make Routing Scale

• Flat (Ethernet) versus Hierarchical (Internet) Addresses
  – All hosts attached to same network have same network address

• Problem: inefficient use of Hierarchical Address Space
  – class C with 2 hosts \( \frac{2}{255} = 0.78\% \) efficient
  – class B with 256 hosts \( \frac{256}{65535} = 0.39\% \) efficient

• Problem: still Too Many Networks
  – routing tables do not scale
    • Big tables make routers expensive
  – route propagation protocols do not scale
Today’s Internet

- Consists of ISP’s (Internet Service Providers) who run AS’s (Autonomous Systems)
- All you need to become an ISP is some address space, an AS number and a peer or two
  - Easier said than done
    - Getting addresses and AS number is the tricky part
    - There are public peering points (MAE East, Central and West)
      - NAP’s run by MCI where peering can take place
    - Most peering points are private
- Number of connections have been doubling for some time – how do we deal with this kind of scaling?
Subnetting - 1985

- Original intent was for network to identify one physical network
  - Lots of small networks are what we actually have – how do we handle this?
- Solution: add another level to address/routing hierarchy: *subnet*
  - Allocate addresses to several physical networks
  - Routers in other ASs route all traffic to network as if it is a single physical network
- *Subnet masks* define variable partition of host part
  - 1’s identify subnet, 0’s identify hosts within the subnet
  - Mechanism for sharing a single network number among multiple networks
- Subnets visible only within a site

<table>
<thead>
<tr>
<th>Network number</th>
<th>Host number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network number</td>
<td>Subnet ID</td>
</tr>
</tbody>
</table>

Class B address

```
11111111111111111111111111111111  00000000
```

Subnet mask (255.255.255.0)

```
Network number Subnet ID Host ID
```

Subnetted address
Subnet Example

Forwarding table at router R1

<table>
<thead>
<tr>
<th>Subnet Number</th>
<th>Subnet Mask</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.96.34.0</td>
<td>255.255.255.128</td>
<td>interface 0</td>
</tr>
<tr>
<td>128.96.34.128</td>
<td>255.255.255.128</td>
<td>interface 1</td>
</tr>
<tr>
<td>128.96.33.0</td>
<td>255.255.255.0</td>
<td>R2</td>
</tr>
</tbody>
</table>
Forwarding Algorithm

D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
  D1 = SubnetMask & D
  if D1 = SubnetNum
    if NextHop is an interface
      deliver datagram directly to D
    else
      deliver datagram to NextHop

• Use a default router if nothing matches
• Not necessary for all 1s in subnet mask to be contiguous
• Can put multiple subnets on one physical network
• Subnets not visible from the rest of the Internet
• This is a simple, toy example!!
Subnets contd.

- Subnetting is not the only way to solve scalability problems
- Additional router support is necessary to include netmask and forwarding functionality
- Non-contiguous netmask numbers can be used
  - They make administration more difficult
- Multiple subnets can reside on a single network
  - Requires routers within the network
- Subnets help solve scalability problems
  - Do not require us to use class B or C address for each physical network
  - Help us to aggregate information
- Chief advantage of IP addresses: routers could keep one entry per network instead of one per destination host
Continued Problems with IPv4 Addresses

• Problem:
  – Potential exhaustion of IPv4 address space (due to inefficiency)
    • Class B network numbers are highly prized
      – Not everyone needs one
    • Lots of class C addresses but no one wants them
  – Growth of back bone routing tables
    • We don’t want lots of small networks since this causes large routing tables
    • Route calculation and management requires high computational overhead

• Solution:
  – Allow addresses assigned to a single entity to span multiple classed prefixes
  – Enhance route aggregation
Supernetting

- Assign block of contiguous network numbers to nearby networks
- Called CIDR: Classless Inter-Domain Routing
  - Breaks rigid boundaries between address classes
  - If ISP needs 16 class C addresses, make them contiguous
    - Eg. 192.4.16 to 192.4.31 enables a 20-bit network number
  - Idea is to enable network number to be any length
  - Collapse multiple addresses assigned to a single AS to one address
- Represent blocks (number of class C networks) with a single pair
  \( \text{first\_network\_address}, \text{count} \)
- Restrict block sizes to powers of 2
- Use a bit mask (CIDR mask) to identify block size
- All routers must understand CIDR addressing
CIDR Addresses

• Identifying a CIDR block requires both an address and a mask
  – Slash notation
  – 128.211.168.0/21 for addresses 128.211.168.0 – 128.211.175.255
    • Here the /21 indicates a 21 bit mask
  – All possible CIDR masks can easily be generated
    • /8, /16, /24 correspond to traditional class A, B, C categories

• IP addresses are now arbitrary integers, not classes

• Raises interesting questions about lookups
  – Routers cannot determine the division between prefix and suffix just by looking at the address
    • Hashing does not work well
    • Interesting lookup algorithms have been developed and analyzed
CIDR – A Couple Details

- ISP’s can further subdivide their blocks of addresses using CIDR
- Some prefixes are reserved for private addresses
  - 10/8, 172.16/12, 192.168/16, 169.254/16
  - These are not routable in the Internet
Address Lookup in IP Routers
Routing Table Lookup

Switch Fabric

Routing Table
Forwarding Decision
Routing Table
Forwarding Decision
Routing Table
Forwarding Decision

Output Scheduling
IPv4 Routing Table Size

Source: Geoff Huston, APNIC
IPv4 Routing Table Size

Source: bgp.potaroo.net, 2013
Routing table lookup: Longest Prefix Match

With CIDR, there can be multiple matches for a destination address in the routing table.

**Longest Prefix Match:** Search for the routing table entry that has the longest match with the prefix of the destination IP address (=Most Specific Router):

1. Search for a match on all 32 bits
2. Search for a match for 31 bits
   ...
32. Search for a match on 0 bits

Needed: Data structures that support a fast longest prefix match lookup!

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/8</td>
<td>R1</td>
</tr>
<tr>
<td>128.143.0.0/16</td>
<td>R2</td>
</tr>
<tr>
<td>128.143.64.0/20</td>
<td>R3</td>
</tr>
<tr>
<td>128.143.192.0/20</td>
<td>R3</td>
</tr>
<tr>
<td>128.143.71.0/24</td>
<td>R4</td>
</tr>
<tr>
<td>128.143.71.55/32</td>
<td>R3</td>
</tr>
<tr>
<td>default</td>
<td>R5</td>
</tr>
</tbody>
</table>

The longest prefix match for 128.143.71.21 is for 24 bits with entry 128.143.71.0/24

Datagram will be sent to R4
IP Address Lookup Algorithms

• The following algorithms are suitable for Longest Prefix Match routing table lookups
  
  – Tries
  – Path-Compressed Tries
  – Disjoint-prefix binary Tries
  – Multibit Tries
  – Binary Search on Prefix
  – Prefix Range Search
IP Address Lookup Algorithms

- The following algorithms are suitable for Longest Prefix Match routing table lookups
  - Tries
  - Path-Compressed Tries
  - Disjoint-prefix binary Tries
  - Multibit Tries
  - Binary Search on Prefix
  - Prefix Range Search
What is a Trie?

• A **trie** is a tree-based data structure for storing strings:
  – There is one **node** for every common **prefix**
  – The strings are stored in extra **leaf** nodes

• Tries can be used to store network prefixes
  – **Note:** Prefixes are not only stored at leaf nodes but also at internal nodes
Binary Trie

- Each leaf contains a possible address
- Prefixes in the table are marked (dark)

Search:
- Traverse the tree according to destination address
- Most recent marked node is the current longest prefix
- Search ends when a leaf node is reached
**Binary Trie**

- **Update:**
  - Search for the new entry
  - Search ends when a leaf node is reached
  - If there is no branch to take, insert new node(s)
Compressed Binary Trie

• **Goal:** Eliminate long sequences of 1-child nodes
  • Path compression $\rightarrow$ collapses 1-child branches

• **Path Compression:**
  – Requires to store additional information with nodes $\rightarrow$ Bit number field is added to node
  – Bit string of prefixes must be explicitly stored at nodes
  • Need to make comparison when searching the tree
Compressed Binary Trie

- **Search: “010110”**
  - Root node: Inspect 1st bit and move left
  - “a” node:
    - Check with prefix of a (”0*”) and find a match
    - Inspect 3rd bit and move left
  - “b” node:
    - Check with prefix of b (”01000*”) and determine that there is no match
    - Search stops. Longest prefix match is with a
Disjoint-Prefix Binary Trie

- Multiple matches in longest prefix rule require backtracking of search
- **Goal:** Transform tree as to avoid multiple matches

**Disjoint prefix:**
- Nodes are split so that there is only one match for each prefix ("Leaf pushing")
- Consequence: Internal nodes do not match with prefixes
- Results:
  - $a$ ($0^*$) is split into: $a_1$ ($00^*$), $a_3$ ($010^*$), $a_2$ ($01001^*$)
  - $d$ ($1^*$) is represented as $d_1$ ($101^*$)

<table>
<thead>
<tr>
<th>Prefixes</th>
<th>0*</th>
<th>01000*</th>
<th>011*</th>
<th>1*</th>
<th>100*</th>
<th>1100*</th>
<th>1101*</th>
<th>1110*</th>
<th>1111*</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

25
Variable-Stride Multibit Trie

- **Goal:** Accelerate lookup by inspecting more than one bit at a time
- “Stride”: number of bits inspected at one time
- With k-bit stride, node has up to $2^k$ child nodes

- **2-bit stride:**
  - 1-bit prefix for $a$ ($0^*$) is split into $00^*$ and $01^*$
  - 1-bit prefix for $d$ ($1^*$) is split into $10^*$ and $11^*$
  - 3-bit prefix for $c$ has been expanded to two nodes
  - Why are the prefixes for $b$ and $e$ not expanded?
Complexity of the Lookup

• Complexity is expressed with $O(.)$ ("big – O") notation:
  – describes an **asymptotic upper bound** for the magnitude of a function in terms of another, usually simpler, function.

• $W$: length of the address (32 bits)
• $N$: number of prefix in the routing table

$O(N)$: growth is linear with $N$
$O(N^2)$: growth is quadratic with $N$
Complexity of the Lookup

- **Bounds are expressed for**
  - **Look-up time:** What is the longest lookup time?
  - **Update time:** How long does it take to change an entry?
  - **Memory:** How much memory is required to store the data structure?

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Lookup</th>
<th>Update</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary trie</td>
<td>$O(W)$</td>
<td>$O(W)$</td>
<td>$O(NW)$</td>
</tr>
<tr>
<td>Path-compressed trie</td>
<td>$O(W)$</td>
<td>$O(W)$</td>
<td>$O(NW)$</td>
</tr>
<tr>
<td>k-stride multibit trie</td>
<td>$O(W/k)$</td>
<td>$O(W/k+2^k)$</td>
<td>$O(2^kNW/k)$</td>
</tr>
</tbody>
</table>