Packet classification

Context

Following weeks:
Now: “Packet” classification @ input ports
Next:
• Forwarding modes across switch
• Discard strategies at points of contention, especially output ports
After:
• Scheduling at points of contention, especially output ports
• Traffic conditioning
• Traffic management

Outline

The classification problem
Types of address (part 1: Classfull)
Classification variations
Exact match classification
Lookup tables
Linked lists
Hashing
CAMs
Types of address (part 2: Classless)
Partial match classification
Exact match techniques revisited
Ternary CAMs
Tries, PARTICIA trees

Packet classification: applications

When “packets” arrive at “switch” inputs, need to use packet fields to look up state information:
• Which output port?
  • Switch input to the correct output. “Correct”:
    • port leads to destination
    • port provides appropriate service, e.g. voice traffic over an ATM network
  • Filtering (no output port): Prevent flow of traffic
    • From certain addresses, e.g. from users who haven’t paid
    • To certain addresses/services, e.g. firewall
• What class of service (within the switch)? “appropriate service” again
• Record usage (billing, conditioning, etc)
Keys: inputs to classification (1)

Packet fields that may be used for classification:
- Source and destination address fields
  - Destination address is the most important
  - But source address may be relevant:
    - for filtering
    - when recording usage
    - for “load balancing” (traffic from multiple sources to one destination
      follow different paths to spread load. Prefer all traffic from one source follow
      the same path to preserve sequence)
    - for bridged networks – learn station locations by observing
      source addresses
  - MAC: address fields are 48b
  - Network layer: address fields are 32b (IPv4) or 128b (IPv6)
- Virtual Circuit Identifiers: 16b for ATM (8 or 12b VPI), 12b VLAN
  ID
- Fields indicating required Class of Service:
  - 8b: Type of Service/Diffserv (IPv4), Traffic Class (IPv6)
  - IPv6: 20b Flow Label
  - 8b: Protocol (IPv4), Next Header (IPv6), e.g. TCP, UDP or ICMP
  - 16b: Transport: source and destination ports
    e.g. low delay and throughput for telnet (TCP port 23)
    high delay and high throughput for FTP data (TCP port 20)

⇒ key width $K$. Classification is simple for small $K$ (e.g. ≤ 32b), harder
for larger $K$ (e.g. 48b)

Characteristics of keys

- Values may be correlated
  - e.g. LAN with many NICs bought in bulk from one vendor:
    MAC addresses have same 24 OUI bits (one vendor), and
    similar (perhaps sequential) less-significant bits
  - deal with hash collisions
- Temporal locality: Keys that have been queried recently will
  likely be queried again soon ⇒ caching (trees, small CAMs)
- Some keys will be queried more often than others, e.g. a server
  carrying a high load ⇒ optimize for the common case

OUI = Organizationally Unique Identifier

Sizes of databases

Databases tend to be sparse (few entries, $D$, relative to keyspace $2^K$)
- Bridge $D=1000$ (1000/2^{48} = 1 in 300 billion)
- Backbone router: $D=100k$ (41,578/2^{32} = 1 in 100,000)

<table>
<thead>
<tr>
<th>Site</th>
<th>Entries ($D$)</th>
<th>Next hops ($&lt;2^5$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac East</td>
<td>39,819</td>
<td>56</td>
</tr>
<tr>
<td>Mac West</td>
<td>14,618</td>
<td>55</td>
</tr>
<tr>
<td>AADS</td>
<td>20,299</td>
<td>19</td>
</tr>
<tr>
<td>Pac Bell</td>
<td>20,611</td>
<td>3</td>
</tr>
<tr>
<td>FUNET</td>
<td>41,578</td>
<td>20</td>
</tr>
</tbody>
</table>

- e.g. 56 possible next hops (output ports) ⇒ store in $E=6$ bits
Aggregation of database entries

If multiple addresses are associated with the same data then we can aggregate them:
- Matching the key to either address is sufficient to access the data.
- Only need one database entry for the group of addresses.

If these addresses are numerically adjacent, then they can be summarised by a broader range of addresses,
- e.g. phone exchange:
  (02) 9381 ..., (02) 9382 ..., (02) 9389 ... all go to Sydney
  ⇒ Melbourne phone exchange can summarise this as all (02) 938 ... calls (indeed, all (02) calls) go to Sydney

Aggregation
- reduces number of entries, but
- complicates classification through variable-length labels

We’ll return to this in the context of Classless Interdomain Routing

Classification problem: Parameters

QOS router example

Use $K$ bit key

$K=48$ (32b IP DA + 16b dest port #)

to select one of $D$ entries in database

$D=1000$ (known addresses)

read out $E$ bits of data from entry

$E=4$ ($2^4=16$ interfaces on router)

Evaluating classifiers

Performance figures:
- **Query time**: Time to search the database.
  - Shorter is better
- **Want result in a predictable time – hardware processes**
- **Update time**: Time to update the database.
  - Would prefer fast updates to the table (slow updates may be acceptable in a firewall)

Cost (e.g. RAM size) for given capacity ($K, D, E$)
Outline
The classification problem
Types of address (part 1: Classfull)

Addressing at various layers
Types of identifiers:
• Names: Textual, and hierarchical, e.g. subjects.ee.unsw.edu.au
• Identify computers (or services, e.g. yahoo.com)
• User friendly ⇒ Common at application layer
• Addresses: Compact binary, generally of fixed length.
• Generally identify network interfaces.
• A node may have multiple network interfaces.
• Protocol friendly ⇒ Common at transport layer and below

We’ll concentrate on addresses, but the classification principles also apply to names (e.g. peer-to-peer file sharing: hash file names to point to the machine that stores that file, in a way that evenly distributes files amongst available machines)

Examples of “addresses”
• Internet Protocol addresses: (32b for IPv4, 128b IPv6)
• MAC addresses (48b)
• Protocol and port identifiers (16b for TCP/UDP): Distinguish different processes that may share a network interface.
While they are not called “addresses”, their functionality is similar to that of other addresses.

Hierarchy in addresses
MAC addresses: Hierarchy in allocation
$ ifconfig
eth0 Link encap:Ethernet HWaddr 00:03:47:AA:BB:CC
Hierarchy only eases allocation: vendor/device
Once product is sold, it may be located anywhere ⇒ hierarchy doesn’t indicate location ⇒ ‘non-routable’
IP addresses: Hierarchy in allocation and in location.
Hierarchical: network/subnetwork/host
$ host subjects.ee.unsw.edu.au
subjects.ee.unsw.edu.au is an alias for alpha400.ee.unsw.edu.au
alpha400.ee.unsw.edu.au has address 149.171.36.48
Hierarchy suggests topological locality
⇒ IP addresses can be aggregated; addresses are “routable”.
Structure of IEEE MAC addresses

Roughly divided into two 24-bit halves:

- 24b **Organizationally Unique Identifier (OUI)**
  - Identifies which organization (manufacturer) a range of addresses has been assigned to.
  - Least significant 2 bits of first byte indicate:
    - least significant: 1=group/0=individual (multicast/unicast)
    - 2nd least significant: 0=global/1=local (OUI assigned by IEEE/assigned locally by network admin)

- 24b **“extension identifier”**
  - Assigned by the organization to differentiate individual devices that it manufactures.

Hierarchy only eases allocation: vendor/device
Once product is sold, it may be located anywhere ⇒ hierarchy doesn’t indicate location ⇒ ‘non-routable’

<table>
<thead>
<tr>
<th>OUI</th>
<th>extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:03:47:AA:BB:CC</td>
<td></td>
</tr>
</tbody>
</table>

0 = individual

\% 4 = 0 or 1 = global

Hierarchy in IP address allocation

Internet Corporation for Assigned Names and Numbers (ICANN)

- Asia Pacific
- American Registry for Internet Numbers
- Réseaux IP Européens

- U. New South Wales
- U. Sydney
  - 149.171...., 129.94....
  - 129.78....

- School of EE, School of CSE, School of Physics
  - 149.171.92...
  - 129.94.242...
  - 129.94.162...

Individual computers

IP addresses

- IPv4 has 32b addresses
  - Originally each address belonged to a particular class (see next slide)
  - Later: Classless addressing
- IPv6 has 128b addresses

IP address classes

- Class A: 1.0.0.0 to 127.255.255.255
- Class B: 128.0.0.0 to 191.255.255.255, e.g. UNSW: 129.94.*.* and 149.171.*.*
- Class C: 192.0.0.0 to 223.255.255.255
- Class D: 224.0.0.0 to 239.255.255.255, multicast address
- Class E: 240.0.0.0 to 255.255.255.255, reserved

Copyright © 2005 Tim Moors
Outline

Exact match classification

Lookup tables

Linked lists

Hashing

CAMs

Classification variations

Do all stored rules apply to the same set of key digits?

Yes is the simplest case, e.g. MAC bridges

No can be more complicated, e.g.

- IPv4 classful addresses: limited set of key digits
- IP classless addresses: Longest prefix matching
- Firewalls: Different rules may apply to different fields

Can a key match multiple rules?

No is the simplest case.

Yes with Longest prefix matching and exceptions

Student classification: problem

How can a lecturer rapidly look up student data of \( E \) bits?

e.g. to record result after marking exam (\( E = 3 \) bits)

Each student is identified by a 7 digit student number (\( K = 7 \) digits)

- \( 10^7 \) possible numbers
- only 78 students in class (\( D = 78 \))
- numbers of students in class are correlated, since they are likely to have enrolled at similar times and numbers are allocated sequentially over time (e.g. many beginning 310...)

Manually classifying students

How might a lecturer do it?

1. Sort list of student IDs into numerical order
   - Sacrifices ease of update in order to expedite search

2. Search through list to find match
   - Sequential search is slow
   - Binary search is possible
   - In practice, initial searches may be binary, until lecturer gets a “feel” for the space (vague memory of experience), then initial stride can be more accurate (e.g. 222* ⇒ look at top of list)
Automatically classifying students

227203 3114283
232244
237856
039312
039411
047291
075026
067328
068008
070001
070360
072487
072741
073452
074439
078123
079284
079682
079708

A1: Manual approach: List them in order &:
• search sequentially
• binary search

A2: Use student number to index a table in RAM.
Wastes RAM since $78 << 10^8 = 10^7$

A3: Convert student number into smaller number to
index a table, e.g. choose least significant 3 digits. More efficient
with RAM but may have collisions (Hashing)

A4: Special associative memory that can simultaneously
match all stored words with supplied value. (CAMs)

A5: Other data structures (linked lists, tries, etc)

Naïve solution: Single lookup table

Use the key to index a RAM of $2^E$ words of $E$ bits each

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>3104005</td>
<td>1</td>
</tr>
<tr>
<td>3104004</td>
<td>0</td>
</tr>
<tr>
<td>3104883</td>
<td>1</td>
</tr>
<tr>
<td>3104884</td>
<td>0</td>
</tr>
<tr>
<td>3104885</td>
<td>1</td>
</tr>
<tr>
<td>3104886</td>
<td>0</td>
</tr>
<tr>
<td>3105050</td>
<td>1</td>
</tr>
</tbody>
</table>

e.g. 1b data indicates if student is member of class
including students 3104005, 3104883, 3104885, 3105050

Advantage: Simple and fast (one lookup of simple
memory).
Disadvantage: sparsely populated large RAM ($D << 2^E$)
OK when key is small ⇒ label swapping for virtual circuits

Exact-match classification

e.g. for MAC addresses; IP addresses without aggregation
exceptions

Techniques:
• Lookup tables
• Lists
• Hashing
• Content Addressable Memories
• techniques that can classify by partial match

Feasibility of a single lookup table

IPv4: 32b destination address lookup ⇒ feasible RAM size:
Size of RAM is feasible with classfull addresses:
Class A: 7b network number = 128 word memory
Class B: 14b network number = 16K word memory
Class C: 21b network number = 2M word memory
With larger keys (e.g. 48b MAC DAs, IPv4 with fields other
than DA, e.g. SA and class of service, 128b IPv6 DAs,
etc), size becomes infeasible
List of data in RAM

- Saves space: \( D \) words each storing \( K + E \) bits
- Takes time
  - If data is unsorted:
    - Search: To perform a match: average time \( D/2 \)
    - Updating 1: After removing an item from the list either:
      - invalidate entry (matching process must traverse invalid entries)
      - shift entries to fill space (takes on average \( D/2 \))
  - If data is sorted
    - Search: To perform a match using a Binary search: average \( \log_2(D) \)

Linked lists

- Efficient use of space: \( D \) words each storing \( K + E + \log_2(D) \) bits
- Timing:
  - Easy to update the list
  - To search: average time \( D/2 \)
  - But: Easy to relocate item to head of list (remove+add), and expedite future matching if the item is likely to be matched again in the near future, e.g. due to:
    - bursty traffic – many packets in succession going to same dest
    - traffic is focussed on certain destinations, e.g. servers

Hashing

Definition: Mathematical mapping from large value to smaller one (ideally \( K \rightarrow \log_2(D) \)). Smaller value can then be used to directly index a table. If table contains an entry, check if it matches full key.

Issues: Mapping function should minimise collisions ("hash bash") in which multiple large values being used map to the same smaller value.

To add an entry to the table:
1. Check whether the hash of the new entry collides with an existing value.
2a. If not, add new entry directly to table.
2b. If there is a clash, update the existing entry s.t. it points to classifier used to resolve collisions (e.g. linked list or CAM). Add new entry to that classifier.

e.g. Hashing of student numbers

Using 3 least significant digits
1000 entry table:
- 74 entries for unique IDs
- 2 for collisions

Least significant 2 digits of sum of pairs of digits
100 entry table:
- 36 unique entries (about half)
- 18 for collisions (worst: 4)

Compare 7.4% or 36% utilisation with 0.78E-3% for single lookup
Example hash functions

- Choosing a subset of the key bits
e.g. for student numbers or MAC addresses:
  - Least significant bits: good (differentiate individuals)
  - Most significant bits: bad (students: similar for same year; MAC: LAN may have many devices from one manufacturer)

Usually apply a more sophisticated function s.t. hashing process is insensitive to format and it is difficult to find a set of addresses that cause collisions

- Arithmetic operations
  - e.g. add digits in student number
  - more complex operations, e.g. CRC

Let’s try hashing!

- From the course outline: You can gain bonus marks through participation:
  - Contributing useful information to the course
  - Pointers to relevant information
  - Relevant stimulating questions
  - Corrections to lectures
  - Attendance

- Process: Pass the lecturer & show your student ID card.
  He’ll use 2 least significant digits to index a hash table, and give you a bonus mark!

CRCs as hashing functions

CRCs make decent hashing functions for Ethernet switches:

- Scramble output well
- Need to calculate anyhow. Calculate CRC over whole frame, but take intermediate result of calculation (after processing first 6 or 12B of header) for hash function.
- Important keys (DA/SA) are at start of frame, so CRC won’t vary because of preceding fields (unlike 802.11 – hash depends on duration)

<table>
<thead>
<tr>
<th>Destination Address 6B</th>
<th>Source Address 6B</th>
<th>Type 2B</th>
<th>Data 46-1500B</th>
<th>CRC 4B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

c.f. 802.11 frame header:

Let’s try hashing!

- From the course outline: You can gain bonus marks through participation:
  - Contributing useful information to the course
  - Pointers to relevant information
  - Relevant stimulating questions
  - Corrections to lectures
  - Attendance

- Process: Pass the lecturer & show your student ID card.
  He’ll use 2 least significant digits to index a hash table, and give you a bonus mark!

Content Addressable Memories

Each word of memory contains:

- $K$ bits of label storage and comparison logic
- and possibly data.

Matching process:
1. Key is distributed to all words simultaneously.
2. Comparison operations are done in parallel.
3. Words with matching labels generate signal.

Figure shows a CAM with

- 4 words with 3b labels and 1b data
- $D \leq 4$, $K=3$, $E=1$
Processing of match signals:

- **OR** them to determine if there is any match (e.g., packet filtering). CAM may output data associated with word.
- **Encode** to index separate table of data.
- **Arbitrate** (e.g., lowest word) multiple matches (e.g., when searching for empty word, prioritising rules, or partial-match searching).

**Example CAM:** MUAA8K80

- 20ns clock speed;
- 80-bit width with programmable CAM/RAM partition
- For bridges: Auto-learn Internal aging with 9-bit time stamp
- Synchronous port for high speed packet processing
- Asynchronous port for table maintenance

**Package:** 160 PQFP

Recall also, the RTL 8308 chip, used in the D-link DES-1008D switch, has a 128-entry CAM to accommodate hash bashing.

**Are CAMs the ideal classifier?**

Sample specs:

<table>
<thead>
<tr>
<th>Device</th>
<th>Capacity</th>
<th>Clock Speed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Music Semiconductors MUAA8K80</td>
<td>8K x 80b</td>
<td>50</td>
</tr>
<tr>
<td>IDT 75K62100</td>
<td>128K x 72b</td>
<td>133</td>
</tr>
</tbody>
</table>

- High speed: 64B Ethernet frames @ 1Gb/s = 2M frames/s << 50MHz (Although multiple, e.g., 2x32b bus, clock cycles required to feed data in/out)
- Low storage capacity/density
  - Comparison logic for each cell
  - Relatively low density (bits/unit area)
  - e.g., 4x6 transistors for SRAM cell, 11x5 for CAM ⇒ more space/bits for CAMs
  - High power consumption
    - e.g., 2W for 8Mb SRAM @ 200MHz, 7W for 2Mb CAM @ 50MHz
  - Niche memory ⇒ not improved through competitive pressures (aka DRAM)

**Hybrid classifiers**

Try using one fast selective classifier & if that fails, revert to another slower but comprehensive classifier, e.g.,

- Hash then CAM (e.g., in RTL8308 chip used in D-link DES-1008D):
  1. Try hashing first.
  2. Resolve hash bashes by storing colliding labels in a CAM. Small CAM resolves rare hash bashes.
- CAM then trie:
  - Store most recently used labels in a CAM.
    1. Try the CAM first (fast)
    2. If CAM is inconclusive, then use a trie (described shortly) that stores all labels.
  - Common cases are processed fast. Trie (large but slow) deals with exceptional cases.
Outline

Types of address (part 2: Classless)

Problems with classfull addresses

- Free addresses become scarce as Internet becomes popular
  ⇒ 128b for IPv6
- Poor utilisation: Organisation O with 2^4 hosts on one network needs Class B space and uses only 1/256th
  - Could use 2 separate Class C networks, but that increases number of entries in router databases (D)
- Router tables becoming too large (2M Class C nets)

Solution: Classless InterDomain Routing (CIDR)

Classless Inter Domain Routing

CIDR allows network prefixes of arbitrary length
Different way of indicating which bits identify network, and which bits identify host.
- No longer indicated by address class
- Instead, indicated by a separate network mask (=1 in network ID bits)
- Routers exchange masks with other reachability info.

Address: 10010101.10101100.00100100.00110000 (149.171.36.48)
Mask: 11111111 11111110 00000000

CIDR prefix lengths

Usually the mask covers contiguous bits, starting with the first bit, so it can be defined by a prefix length:
⇒ can think of addresses as being dotted decimal/prefix_length e.g.
149.171.36.48/23 = 147.171.36 (net) + 0.48 (host)
149.171.37.48/23 = 147.171.36 (net) + 1.48 (host)
37.48 = 0100101
Prefix length indicates significance of trailing 0s, e.g.
149.171.36.0/24 = 149.171.0010010b.h (b-bit identifying host),
149.171.36.0/25 = 149.171.00100100b.h
Routing protocols exchange network addresses + prefix lengths e.g. assign one organisation (“O”):
- two “Class C” spaces (192.16.32 & 192.16.33)
- but only one router entry of 192.16.32/23

In theory, the bits of the network mask need not be contiguous. However, the RFCs defining classless addressing never defined which rule to use if multiple rules have the same length and matched the same address, leading to possible routing loops with non-contiguous masks. See Perlman p. 349
Address aggregation: Scenario

Router has moderate number of ports (e.g. 4-64) & different networks† may be reachable through same port

* e.g. two organisations
  - O with network 192.16.32/23 through port P
  - Q with network 192.16.34/23 through port P
  - Forward aggregate 192.16.32/22 through port P

* might be able to continue the aggregation with other addresses
  (particularly if similar addresses are located in similar areas, e.g. APNIC allocates similar addresses to Australian companies)

* e.g. forward aggregate 192.16.32/19 through port P

Address aggregation: Implementation

Aggregation reduces the number of router table entries.

Ideally assign topologically-adjacent organisations numerically-adjacent addresses; e.g. APNIC

Organizations may move (e.g. Q shifts headquarters overseas or changes network service provider) ⇒ aggregation exceptions
  - e.g. 192.16.32/19 through port P, except 192.16.34/23 through port R

Search for longest prefix match (aka “best prefix match”): longer prefix is more specific and so prefer to route there (give it priority)

Prioritisation of rules

Classification rules may intersect ⇒ need to prioritize, e.g.:
  - Router: Longest-prefix matching
  - Firewall: Filtering traffic may have priority over sending certain traffic on matching path, e.g.:
    1. Block traffic from network N
    2. Forward traffic to network M through port P

Packet from N to M should be blocked.
Partial matching

Rules may be based on exact match or partial match against key

Partial match may be based on:
- prefix – common, e.g. IP network classification
- selected fields – less common, e.g. gateway/firewall may have multiple rules, e.g. ...

Examples of classification on selected fields

Some depend on SA (e.g. block all from spammer), others don’t care about SA (e.g. block all requests to insecure finger service port 79).

Some depend on Class Of Service (e.g. telnet traffic on port 23 to low delay queue, FTP traffic on port 20 to high throughput queue), others don’t care.

When matching on selected fields, can’t co-locate don’t care fields as a suffix ⇒ can’t use longest-prefix matching techniques.

IP header

TCP header

Partial matching with the preceding classifiers

Tables
- Hard to update – e.g. to install rule with mask that doesn’t care about b bits may have to update $2^b$ table entries.
- No benefit from aggregation

Lists
- OK if place longest prefix first; can’t sort to expedite search
- Still doesn’t scale well to many entries.

Hashing
- Only suitable for exact matches – not longest prefix

CAMs – add ternary functionality …

Ternary CAMs

Ternary comparison: 0, 1, X (“Don’t care” = 0 or 1)

Don’t care bits generally specified as a mask (e.g. 1=“Do care”, 0=“Don’t”)

Key mask
apply to all words; iterative search
e.g. mask = 24 ones, then 23 ones, then 22…

Word mask
word-specific, stored with label

CAMs are good when excluded bits needn’t form a suffix, e.g. gateway
Tries

Trie (from re\text{trie}val):
“a tree where each node corresponds to a string that is defined by the path to that node from the root.”

Rule Prefix

<table>
<thead>
<tr>
<th>Rule Prefix</th>
<th>Bit</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0,0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>2,3</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>4,6</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>5,6</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>7,9</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>8,9</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>10,0</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td>10,0</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>11,0</td>
</tr>
<tr>
<td>j</td>
<td>1</td>
<td>11,0</td>
</tr>
</tbody>
</table>

Application to partial match: Locate “don’t care” at end of key. OK for CIDR and address aggregation, but not for gateway filtering.

Implementing tries in RAM

Each node stored in one word, containing at most 3 ptrs:
- left: next rule if bit is 0
- centre: next rule if bit is 1
- right: rule if this is last matched bit

Start search at node 1, continue until either:
- match leaf node, e.g. 011 matches c
- mismatch leaf node, then use previous best match, e.g. 01001 matches a en route to mismatching b.

Optimising tries

Even if trie uses lots of RAM, most recently used nodes will be stored in (processor’s) cache ⇒ Faster for recently used queries.

- Lengthening the “stride” with multi-bit tries
- Collapsing nonbranching branches

Multi-bit strides

Rather than binary movement at each node (at most 2 child nodes), we could make bigger strides at each node, e.g.:
- decimal: up to 10 children
- more likely: an integer-sized group of bits (e.g. 2 bits = up to 4 children, 3b = up to 8 children, ...)

Terminology: “stride” size = number of bits used to branch out from each node, e.g. stride of 1 = binary

Larger stride:
- reduces number of nodes traversed, increasing speed
- requires larger tables that may be sparsely occupied (with memories increasing in size faster than increasing in speed, wasting some space for number of accesses makes sense)
IP Route Lookup: Multi-bit Tries

Path-compressed tries

Observation: When keys are sparse, many trie nodes have only one descendent, taking many steps to reach a single result.

Solution: Rather than use every bit of key to determine subtree, record in node which bits should be used. At leaves, compare (in parallel) all bits of leaf with key. If mismatch, then use previous best match.

Provides “path compression” by recording only genuine branches.

PATRICIA (a similar path-compression scheme) is the classification technique used with BSD Unix (so claims [Ruiz-Sanchez] -- bonus mark if you can identify the Linux algorithm)

Resources

Relevant sections of Keshav:
• Chapter 10: Addressing
• pp. 176-8: classification (“port mappers”)

