Routing and Scalability in s3fnet

Trie Summary

- Trie 0 (trie0.cc) – S3fNet's original trie
  - No performance gains/penalties
  - CIDR requires address input from longest prefix to shortest (ie ip1, ip2/31, ..., ip100/16, ..., ip1000/1)
- Trie 1 (trie1.cc) – Simple enhancements only
  - Identical to trie0 with slight code optimizations
  - Basis for Tries 2+; should modify THIS not trie0.cc
- Trie 2 (trie2.cc) – Compressed shortest-prefix
  - Stores TrieData at minimum required depth vs depth 32 (or 64 if ipv6 support is ever added)
  - CIDR requires shortest → longest prefix input

Trie Summary

- Trie 3 (trie3.cc) – Unordered prefix
  - Functionally identical to Trie 2 but without the CIDR input-order constraint; suitable for dynamic routing as a result
- Trie 4 (trie4.cc) – Array-backed
  - RouteData stored in global array; tries store indexes
  - TrieDataArray0 & 1 – reference implementations only to show array overhead (1 uses pointers, 0 uses direct memory)
  - TrieDataArray2 – Coalesces duplicate RouteData entries across all hosts' array-backed tries

Using Alternative Trie Models

- Current support only via '#define' in forwardingtable.cc
  - Uses defaults: Trie::SIMPLE (trie1) & RouteCache::SINGLE_ENTRY
  - Array-backed uses TrieDataArray::NAIVE (tda2)
- DML-based Trie & Cache selection support exists in forwardingtable.cc but must be added to ip_session.cc & any other sessions creating forwarding tables

Recommended Use Cases

- Tries
  - Many Hosts, Many Messages – Array-backed (4)
  - Dynamic routing – Unordered-prefix (3)
  - Lower memory usage – prefix (2)
  - Simple (1) otherwise
- Caching
  - Few Messages, Few Hosts – No caching (NONE)
  - Very Many Messages – Associative
  - Single-entry otherwise
- Raw performance data available in triedata.tar.gz

Background & Performance Results

The s3fnet Data Model

- DML files parsed by s3f framework
- test.dml defines nets, hosts, & interfaces

```
total_timeline 1
tick_per_second 6
sim_single_run_time 10
Net [
    Net [ id 1 alignment 0
        host [ id 1
            graph [
            ProtocolSession [ name dummy use "s3f.os.dummy"
            hello_interval 2e-06 hello_peer_nhi 1:2(1) hello_message "hello from host 1:1"
            ]
            ProtocolSession [ name ip use "s3f.os.ip" ]
            ]
            interface [ id 1 _extends .dict.iface ]
            interface [ id 2 _extends .dict.iface ]
            interface [ id 3 _extends .dict.iface ]
        ]
        host [ id 2
            graph [
            ProtocolSession [ name dummy use "s3f.os.dummy"
            hello_interval 2e-06 hello_peer_nhi 1:1(1) hello_message "hello from host 1:2"
            ]
            ProtocolSession [ name ip use "s3f.os.ip" ]
            ]
            interface [ id 1 _extends .dict.iface ]
        ]

    link [ attach 1(1) attach 2(1) min_delay 1e-06 prop_delay 1e-06 ]
    ]
```

The s3fnet Data Model

- test-env.dml defines environment, including IPs
  - Created by dmlenv binary

```
environment_info [
  Net [
    Net [ id 1 prefix 10.10.20.128 prefix_int 168432768 prefix_len 29 ]
    Net [ id 2 prefix 10.10.20.136 prefix_int 168432776 prefix_len 29 ]
    Net [ id 3 prefix 10.10.20.144 prefix_int 168432784 prefix_len 29 ]
    Net [ id 4 prefix 10.10.20.152 prefix_int 168432792 prefix_len 29 ]
  ]
  host [ name no_name uid 0 mapto 1:1(1) alignment 0]
  host [ name no_name uid 1 mapto 1:2(1) alignment 0]
  host [ name no_name uid 2 mapto 2:1(1) alignment 0]
  host [ name no_name uid 3 mapto 2:2(1) alignment 0]
  host [ name no_name uid 4 mapto 3:1(1) alignment 0]

  alignment [
    name 0
    interface [ nhi 1000:1(1) ip 10.10.10.177 delay 0 link_prefix_len 30 peer 1000:1(1)  peer 1000:2(1) ]
    interface [ nhi 1000:1(2) ip 10.10.36.154 delay 0 link_prefix_len 30 peer 999:1(3)  peer 1000:1(2) ]
    interface [ nhi 1000:1(3) ip 10.10.41.130 delay 0 link_prefix_len 30 peer 1:1(2)  peer 1000:1(3) ]
    interface [ nhi 1000:2(1) ip 10.10.10.178 delay 0 link_prefix_len 30 peer 1000:1(1)  peer 1000:2(1) ]
  ]
]
```

The s3fnet Data Model

- test-rt.dml defines routes for static routing
  - Also created by dmlenv binary
  - Uses iterative variant of Dijkstra's algorithm

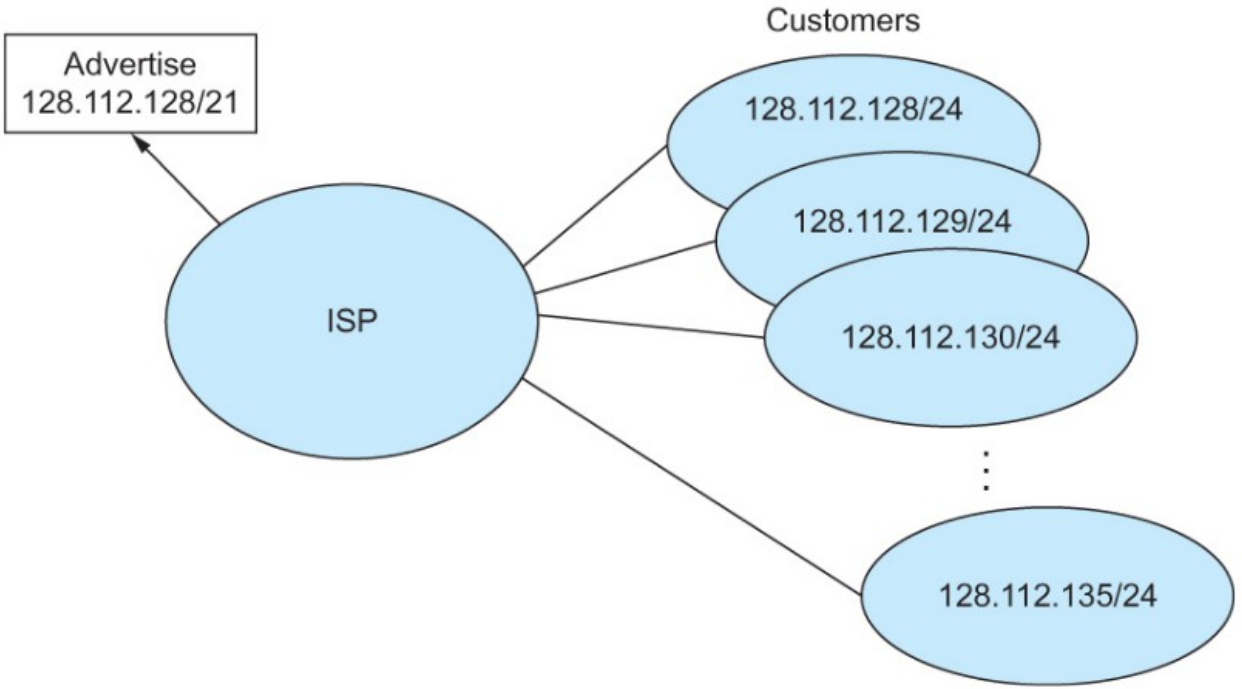
```
#routable destination: 1:1
#routable destination: 1:2
#routable destination: 2:1
forwarding_table [ node_nhi 1:1
  nhi_route [ destip 10.10.33.182/31 interface 2 next_hop :1000:1(3) ]
  nhi_route [ destip 10.10.41.128/30 interface 2 next_hop :1000:1(3) ]
  nhi_route [ destip 10.10.33.184/29 interface 2 next_hop :1000:1(3) ]
  nhi_route [ destip 10.10.20.136/29 interface 3 next_hop :2:1(2) ]
]
forwarding_table [ node_nhi 1:2
  nhi_route [ dest default interface 1 ]
]
forwarding_table [ node_nhi 2:1
  nhi_route [ destip 10.10.33.186/31 interface 2 next_hop :1:1(3) ]
  nhi_route [ destip 10.10.33.188/30 interface 2 next_hop :1:1(3) ]
]
```

The s3fnet Data Model

- test-rt.dml forms a natural bottleneck
  - Filesizes ~ $O(n^2)$ where n = number of hosts
  - 2000 hosts yields ~4 million line test-rt.dml
- Can we compress test-rt.dml without losing any necessary forwarding information?
  - Translate NHI address to IP using env data
  - Merge duplicate routes to shared prefixes (CIDR)
  - Created rtcompress.py to do this
  - Observed a huge improvement! (~61000 lines)

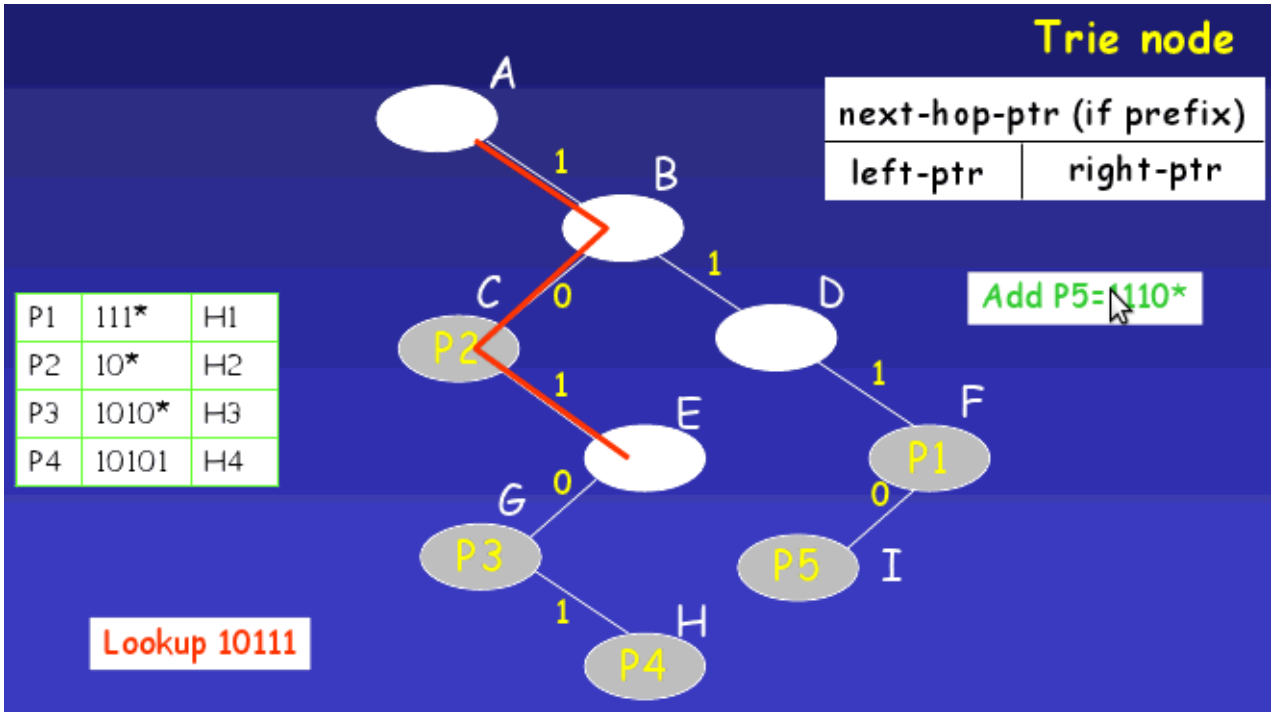
CIDR Addressing Review

- Represent multiple addresses sharing a common prefix with a single address
- Place /X after the prefix where X is the prefix length in bits



Radix (Binary) Tries

- To support CIDR addressing, routers use longest-match policy
- Effective to lookup routes via binary Tries
- s3fnet already has a Trie implementation



Scalability of test-rt.dml

- 2000 hosts:
  - Loading takes >120s; uses ~4.3 GB of RAM
  - With CIDR, loading takes ~3s; uses ~119 MB
  - dmlenv takes ~10s; uses ~340 MB
- 4000 hosts:
  - Loading takes >3000s; uses >16 GB of RAM/swap
  - With CIDR, loading takes ~9s; uses ~238 MB
  - dmlenv takes ~49s; uses ~1.3 GB
- Cannot create test cases for >8000 hosts!

  (Tests run with a 3 GHz 64-bit CPU, 12 GB RAM, 12 GB swap)

  (rtcompress.py takes ~140/600 s for 2000/4000 hosts; mem ~CIDR s3fnet)

Improved test-rt.dml Generation

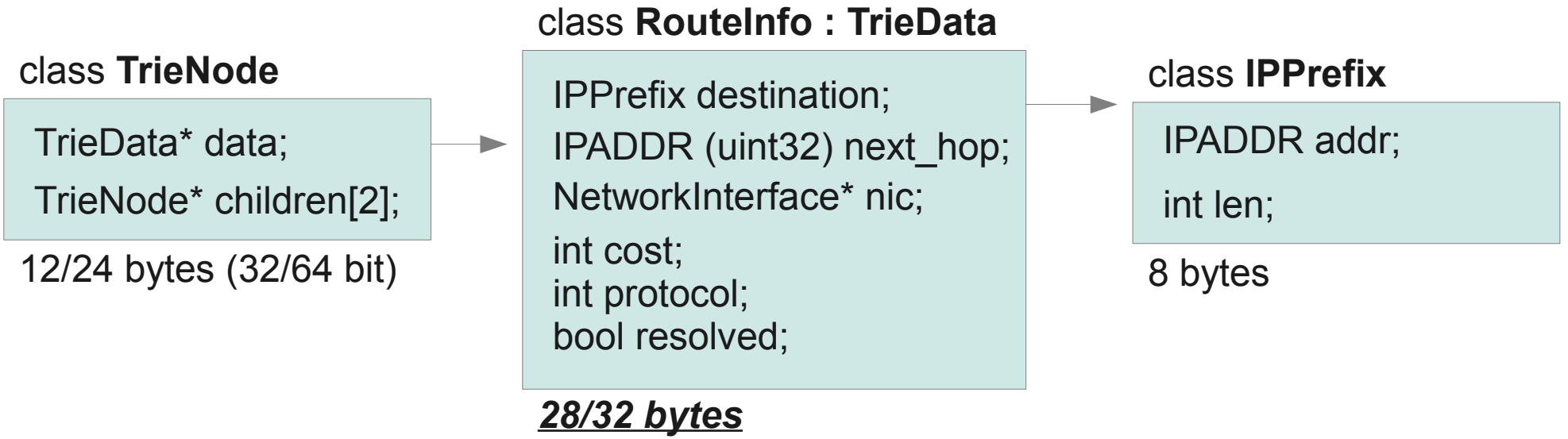
- dmlenv becomes a bottleneck
  - time/memory both grow as $O(n^2)$
- rtcompress.py slow (not a permanent solution!)
- Integrate CIDR generation into dmlenv
  - Requires modifications to s3fnet's Trie structure
  - Requires dijkstra results → will slow down dml creation
  - Adds explicit dependence on test-env.dml
- Change structs & algorithm(s) used in dmlenv
  - Flloyd-Warshall: faster & < 8GB for < 65535 hosts
  - Iterative Dijkstra (repeat computation to save mem)

Questions

- How much modification does s3fnet's Trie require for use in dmlenv?
- Why did CIDR lower memory usage so much?
- Can we improve loading time even more?
- Can we improve routing time *after* loading?
- Can we improve routing memory usage further?

s3fnet's Trie

- s3fnet's ForwardingTable extends its Trie implementation (but it has problems)
- ALWAYS depth 32 (unless loaded otherwise)
  - Cannot do prefixing itself, even though it supports it!
  - Causes additional unnecessary structures
  - Structures already need improvement!



Improving Routing Time

- Caching
- Varied Trie structures
  - Proper prefix Tries
  - Array-backed indices vs data in Trie
- Dummy protocol with n*1000 nets, 2 hosts/net
- Tested for 2000, 4000, & 8000 hosts
- Tested for 0, 1e5, 1e6, & 1e7 messages
- Most testing used CIDR dml optimization
- Gathered data for 10 repetitions (1 for 1e7 or non-CIDR)

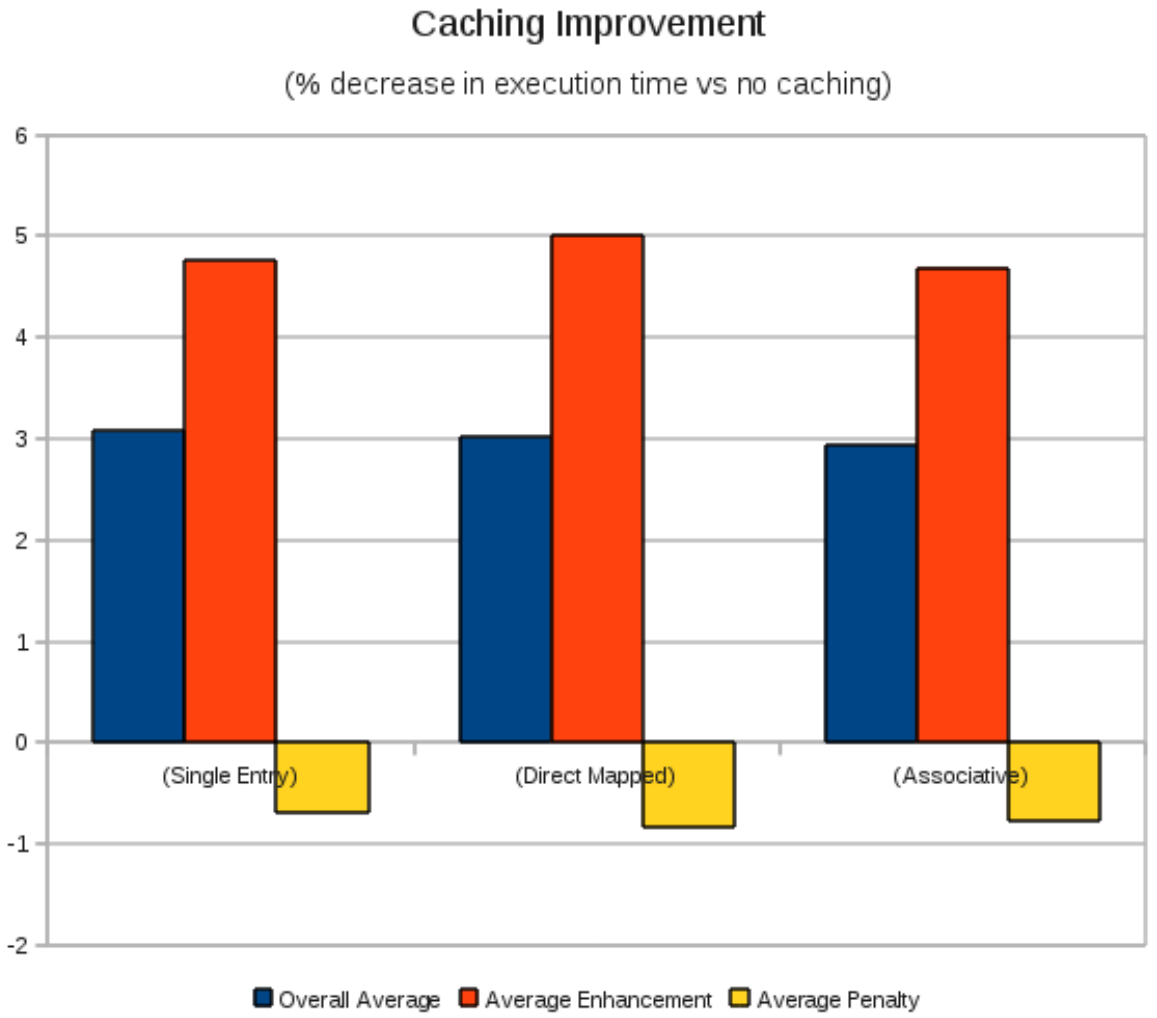
Improving Routing Memory Usage

- Easy to determine caching memory overhead
- Static routing → Only # of hosts matter
  - Tested for 2000, 4000, & 8000 hosts
- Mem usage measured in KB
  - Did not change across any early repetitions
  - Settled on a single test/config
  - Hard to script → not fully evaluated

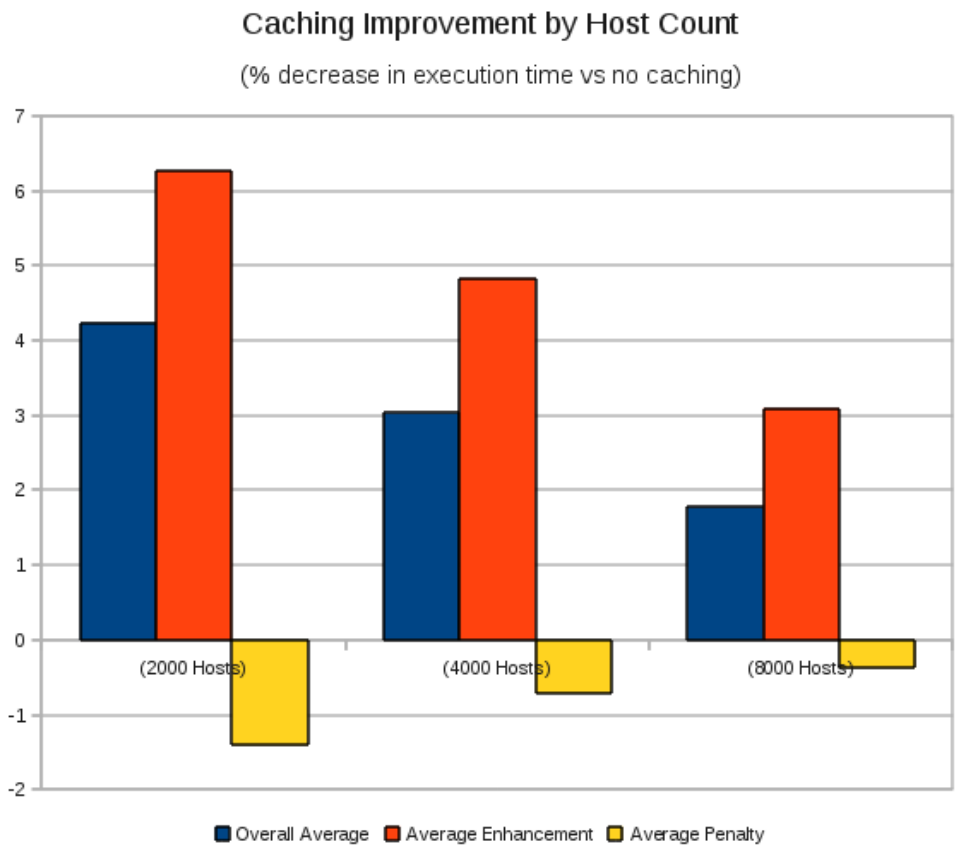
Cache Types

- None
- Single entry
- Direct mapped
  - 256 entry, mapped from last byte of IP
  - Configurable size
- Associative
  - Identical to direct-mapped but 2-way associative
  - Configurable size & associativity

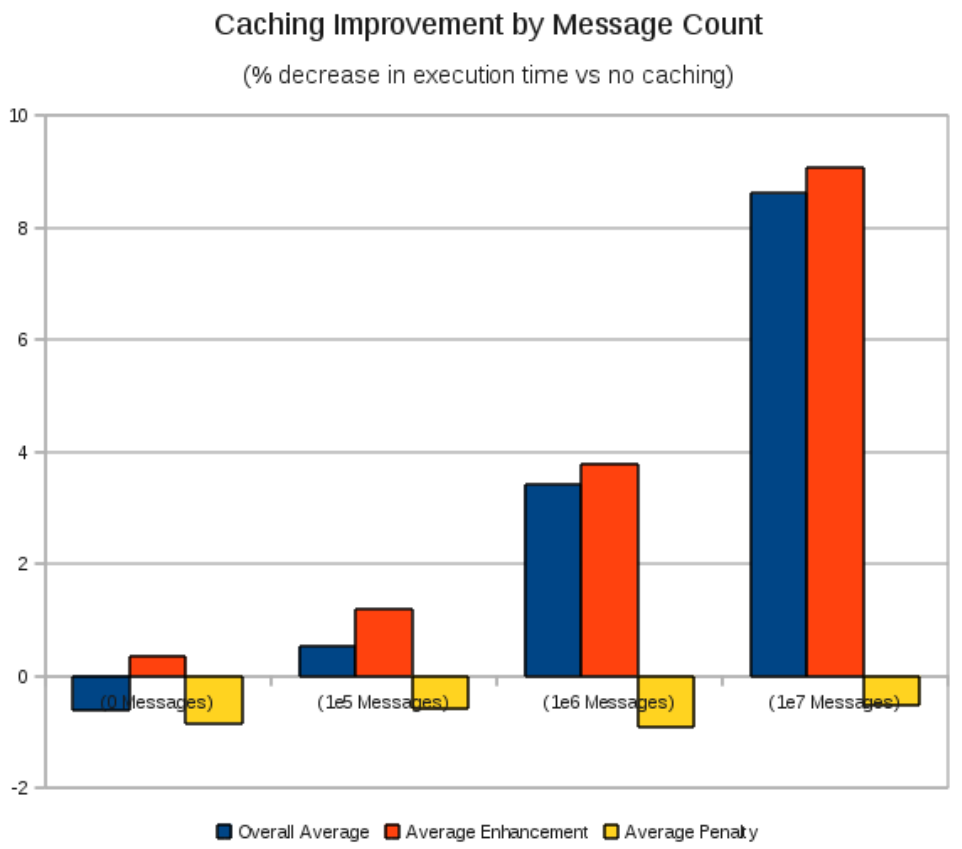
Caching Results



Caching Results



Single-entry caching showed minimal performance loss as hosts increased.



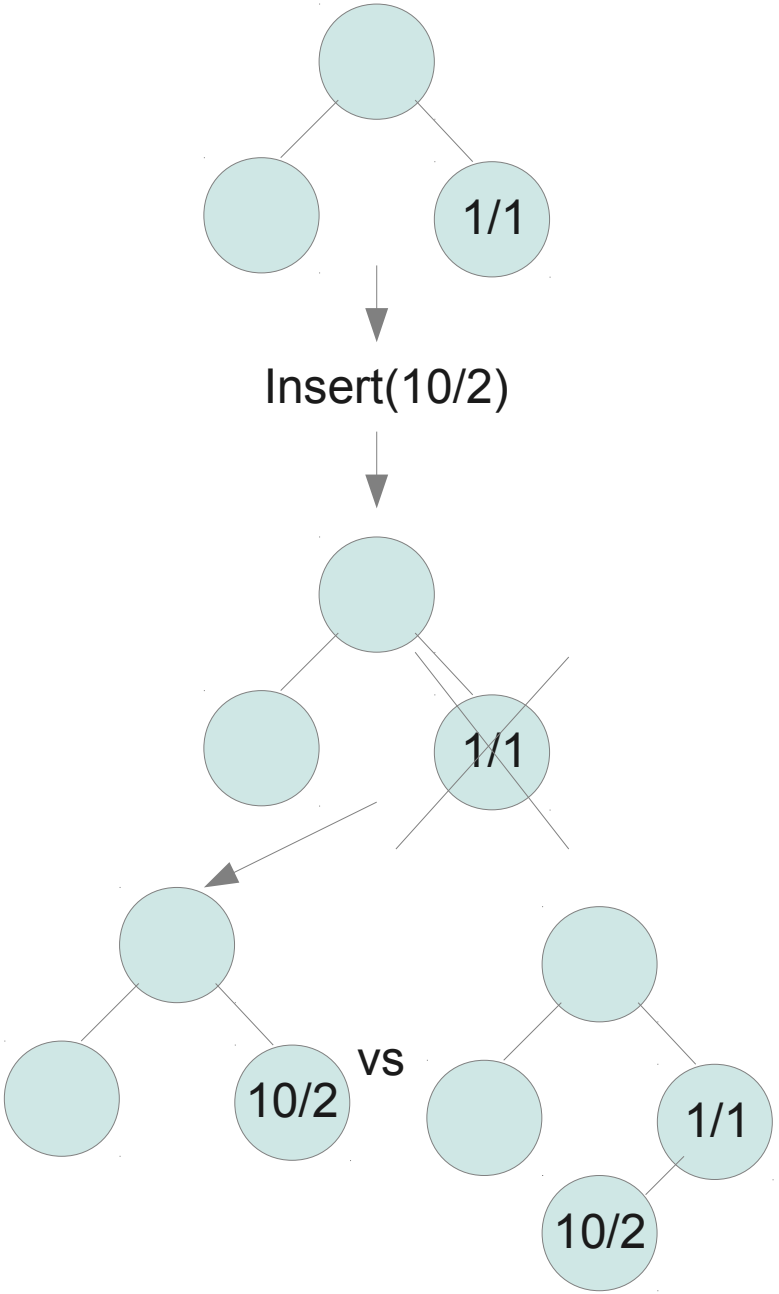
Direct-mapped caching performed best for 1e6+ messages. Associative *may* perform better for 1e8+.

Simple Enhanced Tries (Trie1)

- Slightly optimizes the lookup function
- Adds ability to clean up empty nodes
  - Requires slight overhead increase to the search() utility method
  - We don't remove Trie nodes in static routing, so this doesn't matter for the current simulator
- Designed as a code-base for all other Trie implementations.
- Original implementation (Trie0) kept for comparison.

False-Duplicate Prefix Problem

- Problem:
  - The search method always returns the longest match from the Trie, even if data is not the same.
  - This results in false 'duplicates' if short prefixes are loaded before other routes sharing that short prefix.
- Solution:
  - CIDR routes must be ordered from longest to shortest prefix (w/31,z/30,...,y/2,z/1)
- Trie0 and Trie1 both suffer from this

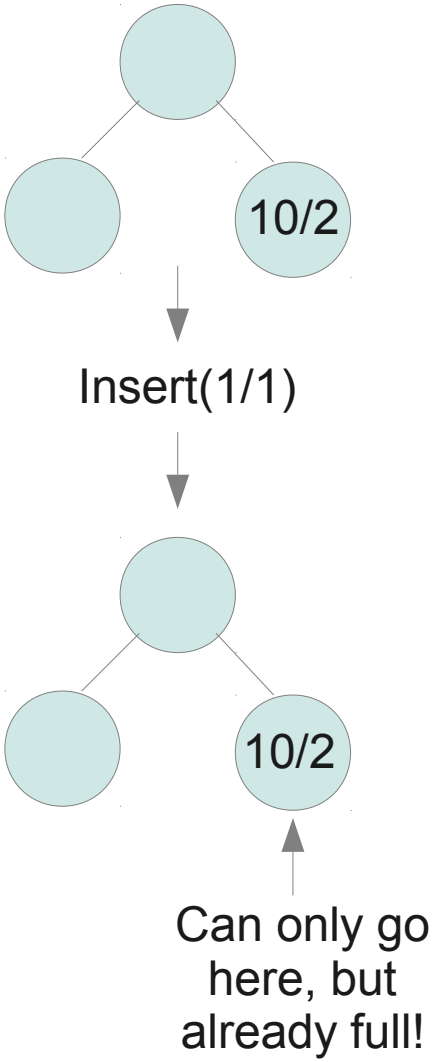


Prefix Tries (Trie2)

- Less memory usage but adds light overhead
- Modifies insert() substantially
  - Data is stored at first available node on its path
  - Eliminates depth 32 guarantee
- Guarantees that all non-null nodes have data
- Usable with dynamic routing (somewhat)
- Introduces a new problem

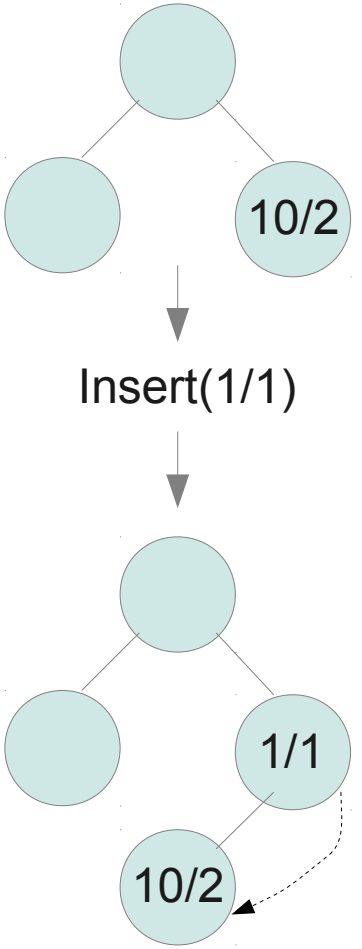
Depth-Saturation Problem

- To solve the False-Duplicate problem with Trie0 and Trie1, we ordered by decreasing prefix length.
- Problem:
  - Now, these long prefixes are added to the top of the Trie.
  - When short prefixes are added, all possible spots they can fill are already full and a duplicate route is registered.
- Solution:
  - Order input by *increasing* prefix length
  - Alternative: Trie3
- This ONLY affects Trie2



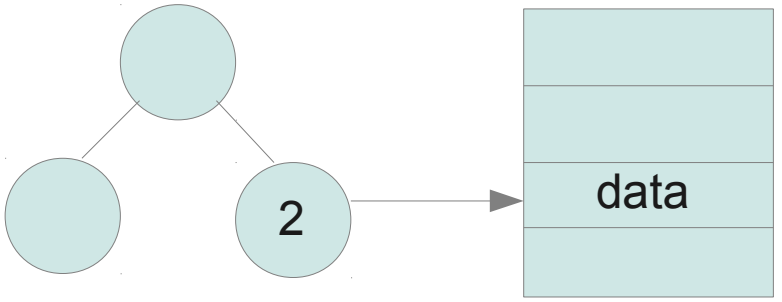
Unordered Prefix Tries (Trie3)

- Less memory usage & no loading constraints but potentially high insertion overhead
- Fully compatible with dynamic routing (hopefully)
- Modifies insert() to propagate nodes downward if a node's max depth is saturated.
- Nodes must maintain their keys to know what propagations are valid
- Not extensively tested with random insertions
  - May still have problematic cases



Array-backed Indexed Tries (Trie4)

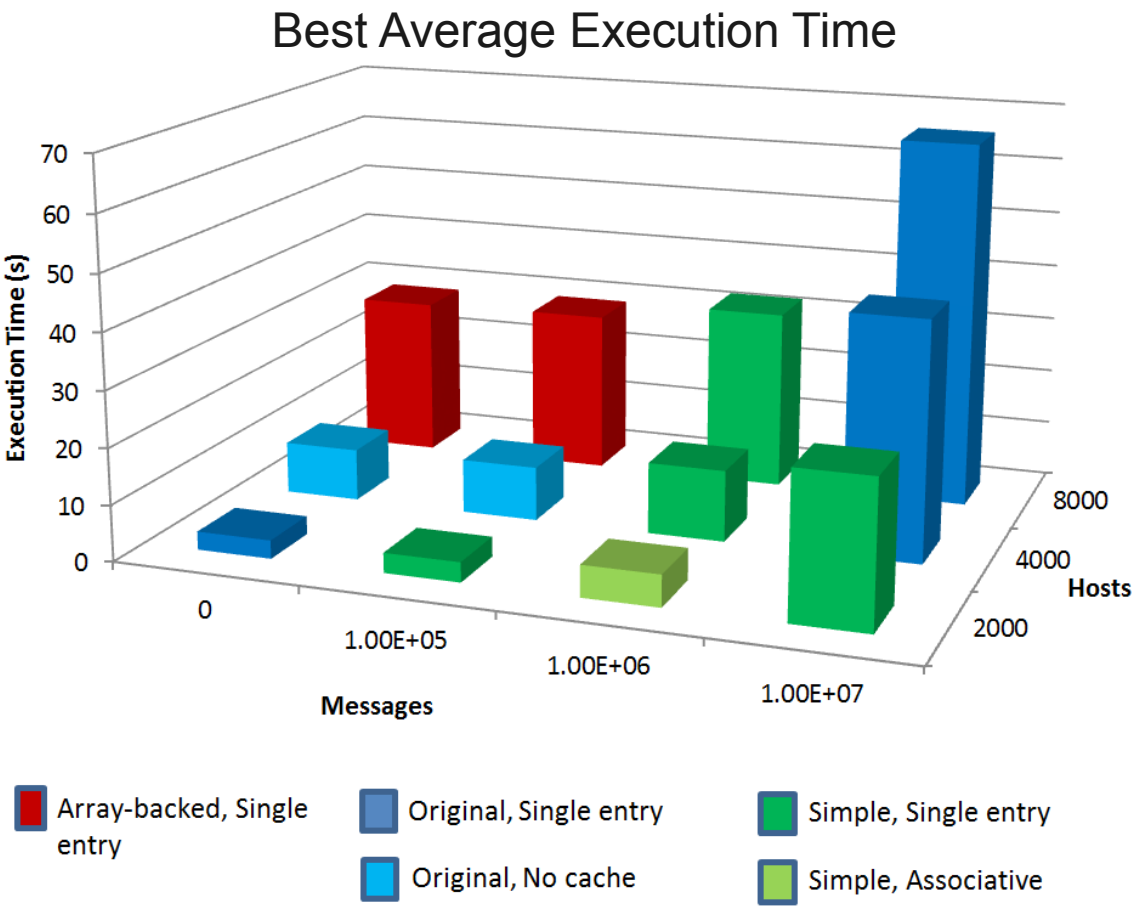
- Prefix Tries compress routing entries to similar destinations, but what about to identical next_hops (duplicate RouteInfo entries)?
- Create an array of RouteInfo entries
- Trie stores index into the array (vs actual data)
- Trie4 is based off Trie1
  - Concept is extendable to all previous Tries



Types of Arrays

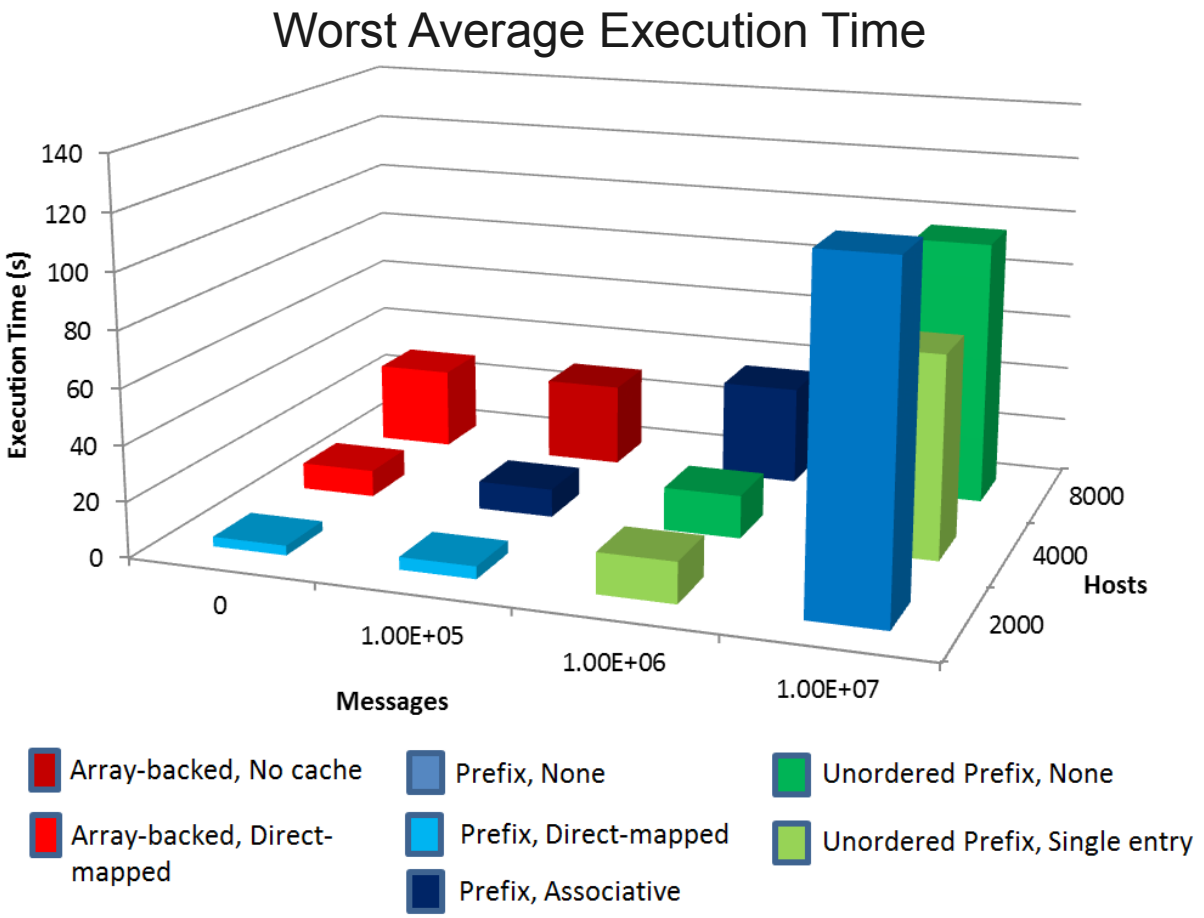
- The main problem with Indexed Tries becomes determining if there is already duplicate data in the array on insert()
- Naïve implementation: search all elements (tda2)
- Better: Use hashing (or at least direct-mapping)
  - Will need an overflow mechanism for collisions
  - May use associativity
  - To gain speed at the cost of some memory, ignore any colliding route (don't check overflow – just add to overflow)
- Also created uncoalesced arrays to measure overhead
  - tda0 directly reads and writes RouteInfo
  - tda1 is effectively a map, storing pointers to RouteInfo
  - Despite the extra memory access, tda1 performed slightly better than tda0 overall and is ***much*** easier to work with.

Execution Time Results



- Loading 2000 hosts ~ sending 1e5-1e6 messages
- Loading 4000-6000 hosts ~ sending 1e6-1e7 messages
- Above this equivalence point, implementation is key

Execution Time Results



- For constant hosts, as messages increase, the penalty for using non-ideal Tries/Caching decreases
  - Gap between best and worst is higher, but a lower percentage of total execution time

Memory Usage Results

- Recap:
  - 2000 hosts → ~4.3GB normal, ~119 MB CIDR
  - 4000 hosts → ~16 GB normal, ~238 MB CIDR
  - 8000 hosts → Not runnable, ~478 MB
- Memory measurements inconsistent
- Tests do not show less memory usage with Prefix Trie Implementations. Why?:
  - RouteInfo structs (big, unique, hard to change)

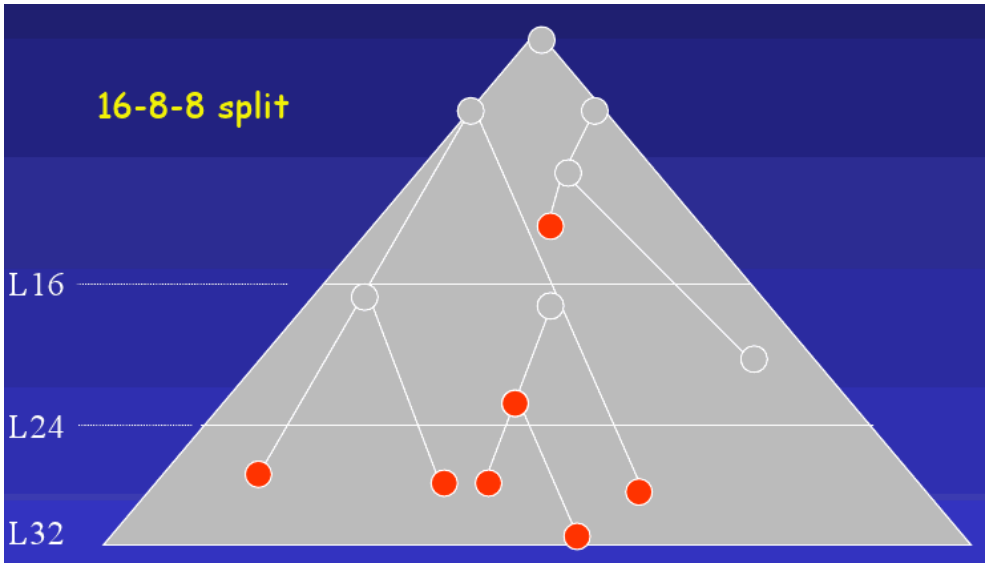
Untested Trie Concepts

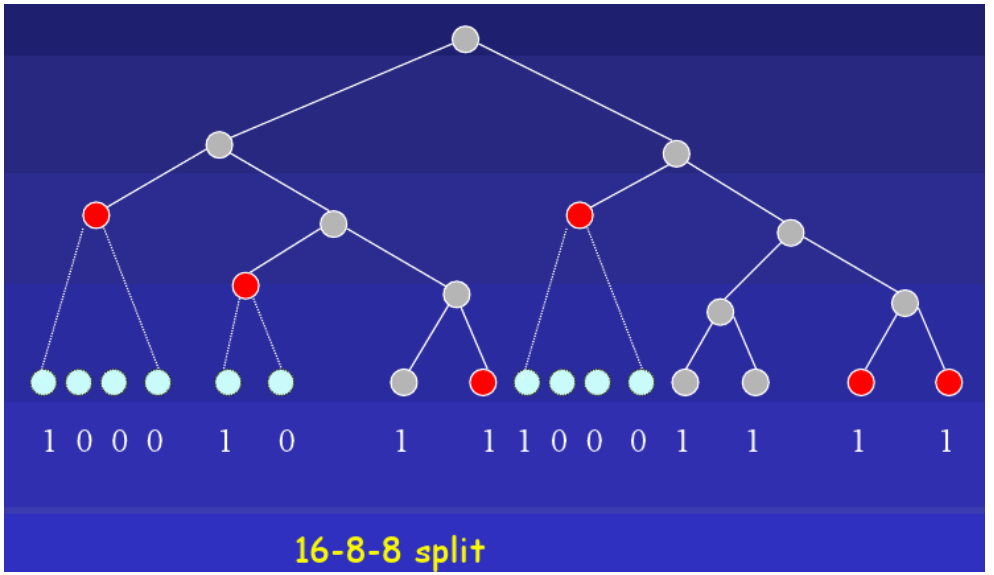
- Array Types:
  - Direct-mapped (pseudo-hashed)
  - Associative
  - Static (shared) array(s) across multiple(/all) hosts
- Trie Types:
  - Separate leaf nodes (no pointers)
  - Prefix/Unordered-Prefix with Array
  - Prefix/Unordered-Prefix with Static Array
  - Lulea Trie (ideal for static routing, but patented...)

Visit http://klamath.stanford.edu/~pankaj/talks/hoti_tutorial.ppt for ***WAY*** more on Tries.

Lulea Trie

- Split Trie into 3 levels
- Use binary 'existence' string to map into second level
- Extremely small, fast data structure
- Insertion complicated
  - Probably not good for dynamic routing!





Original paper:
http://www.cs.cmu.edu/~dga/15-744/S07/papers/D+97.pdf

Summary

- Main bottleneck is dml generation
- CIDR is scalable, but increases generation time
- dmlenv must be modified for 10000+ hosts
- Caching is effective, but only gains ~10% time for 1e6+ messages (and uses more mem)
- Prefix Tries save memory and may support dynamic routing, but are comparatively SLOW
- RouteInfo structure needs to be pruned