Analysis of Range based Data Structures for Software Based IP routing tables.

Motivation

- Classful IP addressing system - fixed and rigid allocation based on classes
- CIDR - uses prefix notation
- VLSM - specific implementation of CIDR, wherein further division of IP address blocks into smaller subnets is allowed

★ Dynamically allocate and deallocate subnets and host IP addresses efficiently with routing capabilities.

Methodology

- Compared four approaches -
  - Array
  - Balanced Binary Tree
  - Vam Emde Boas Tree
  - Hashed Vam Emde Boas Tree

- Hybrid of DHCP and routing

- Usage of IP address spaces more efficiently

Tools and Technologies Used

- Programming Language - C++
- Mimicked DHCP Server with routing capability - utilized CPP server to host HTTP application
- Postman

Key Functionalities and Outcomes

- Dynamic allocation and deallocation of Subnets and Host IP addresses
- Minimal to no wastage of IP addresses - Optimal usage of IP spaces
- Renewal of Host IP addresses
- Routing

Scenario	Results (Performed best)
Insertion of 260K Subnets with 1000 capacity each	Array Implementation
Deletion of 130K Subnets at odd positions	Hashed Van Emde Boas Tree
Querying the Subnet IP for 130K Hosts (1 host per subnet)	Array Implementation
Re-inserting 130K subnets again	Hashed Van Emde Boas Tree

DEMO

Before assigning Subnets

Subnets (Assigned subnets)			
Subnet	**Start IP**	**Capacity**	**Hosts**

Free slots (Unassigned IP blocks)	
Start IP	**Capacity**
0	Total capacity (2 ^ 28)

After assigning 3 subnets and 1 host each

Subnets (Assigned subnets)			
Subnet	**Start IP**	**Capacity**	**Hosts**
1	0 (0 - 99)	100	MAC: 1 IP: 0.0.0.1
2	100 (100 - 199)	100	MAC: 101 IP: 0.0.0.101
3	200 (200 - 299)	100	MAC: 201 IP: 0.0.0.201

Free slots (Unassigned IP blocks)	
Start IP	**Capacity**
300	Total - 300

After deleting a subnet - leaving a free slot

Subnets (Assigned subnets)			
Subnet	**Start IP**	**Capacity**	**Hosts**
1	0 (0 - 99)	100	MAC: 1 IP: 0.0.0.1
3	200 (200 - 299)	100	MAC: 201 IP: 0.0.0.201

Free slots (Unassigned IP blocks)	
Start IP	**Capacity**
100	100
300	Total - 300

New subnet added to the free slot

Subnets (Assigned subnets)			
Subnet	**Start IP**	**Capacity**	**Hosts**
1	0 (0 - 99)	100	MAC: 1 IP: 0.0.0.1
4	100 (100 - 149)	50	
3	200 (200 - 299)	100	MAC: 201 IP: 0.0.0.201

Free slots (Unassigned IP blocks)	
Start IP	**Capacity**
150	50
300	Total - 300

After optimizing the free slot

Subnets (Assigned subnets)			
Subnet	**Start IP**	**Capacity**	**Hosts**
1	0 (0 - 99)	100	MAC: 1 IP: 0.0.0.1
3	100 (100 - 199)	100	MAC: 201 IP: 0.0.0.101
4	200 (200 - 249)	50	

Free slots (Unassigned IP blocks)	
Start IP	**Capacity**
250	Total - 250

Implementation	Implementation DS	Usage
Array Implementation	Data Structure 1 : Sorted Array Map. Key – Subnet IP, Value – Subnet Capacity Data Structure 2 : Sorted Array Map . Key – Subnet Capacity, Value – List of Subnets with that Capacity.	DS 1 - Used for inserting subnets and finding the subnet IP of a particular host. DS2 - Used to store free slots available for allocation
Balanced Binary Tree Implementation	Data Structure 1 : Tree Map. Key – Subnet IP, Value – Subnet Capacity Data Structure 2 : Sorted Array Map . Key – Subnet Capacity, Value – List of Subnets with that Capacity.	DS 1 - Used for inserting subnets and finding the subnet IP of a particular host. DS2 - Used to store free slots available for allocation.
Van Emde Boas Tree Implementation	Data Structure 1 :Veb Tree Map. Key – Subnet IP, Value – Subnet Capacity Data Structure 2 : Veb Tree Map. Key – Subnet Capacity, Value – List of Subnets with that Capacity.	DS 1 - Used for inserting subnets and finding the subnet IP of a particular host. DS2 - Used to store free slots available for allocation.
Van Emde Boas Tree with HashMap	Data Structure 1 : Hashed Veb Tree Map. Key – Subnet IP, Value – Subnet Capacity Data Structure 2 : Hashed Veb Tree Map. Key – Subnet Capacity, Value – List of Subnets with that Capacity.	DS 1 - Used for inserting subnets and finding the subnet IP of a particular host. DS2 - Used to store free slots available for allocation.

Implementation details

- The system was implemented in C++
- For Implementation 1 and Implementation 2, Data Structures where available in the C++ standard library
- For Implementation 3 and 4 , We made implemented the Data Structures from scratch

Benchmarking

- Benchmarking was done in a Macbook M2 with 16GB of RAM

Scenario	Time taken for Array Impl (ms)	Time taken for Tree Impl (ms)	Time Taken for Veb Tree impl (ms)	Time taken for Hashed Veb Tree Impl (ms)
Insertion of 260 k Subnets of capacity 1000 each on an empty state.	**45**	**67**	**780**	**94**
Deletion of 130 k subnets at odd positions . 1st subnet , 3rd subnet so on are deleted. (1st , 3rd refers to order of insertion	**10438**	**1575**	**38**	**43**
Querying the subnet Ip for 130 k Hosts. 1 host per each subnets.	**4**	**5**	**65**	**16**
Re-inserting 130 k subnets again	**11323**	**1345**	**356**	**13**

Conclusions

- Array Implementation performs well on Subnet queries for hosts(i.e. Finding the subnet that a host belongs to). But deletions and re-insertions are considerably slower.
- Hashed VEB Tree Implementation is slower for Subnet queries but faster for deletions and re-insertions.
- BST Implementation is slightly slower than Array Impl for subnet queries, but faster than Array Impl for deletions and re-insertions, but still slower than Hashed VEB.
- VEB Tree Implementation takes a considerable amount of initial startup time and uses too much memory.
- Hashed VEB Tree Implementation has good overall performance and is suited for dynamic scenarios.

Future Scopes

- Investigating approaches to reduce memory usage of the above approaches
- Investigating bisection strategy for Subnet allocation. Gives more flexibility to grow subnet size without changing IP addresses
- Testing this strategy using SDNs and NFVs

Thanks!