Packet forwarding

- How do routers process IP packets
- How do they forward packets
- Assigned reading
  - Fast and Scalable schemes for the IP address Lookup Problem

Forwarding vs. Routing

- Forwarding: the process of moving packets from input to output
  - The forwarding table Lookup.
  - How to populate the lookup table?
- Routing: process by which the forwarding table is built and maintained
  - One or more routing protocols
  - Procedures (algorithms) to convert routing info to forwarding table.

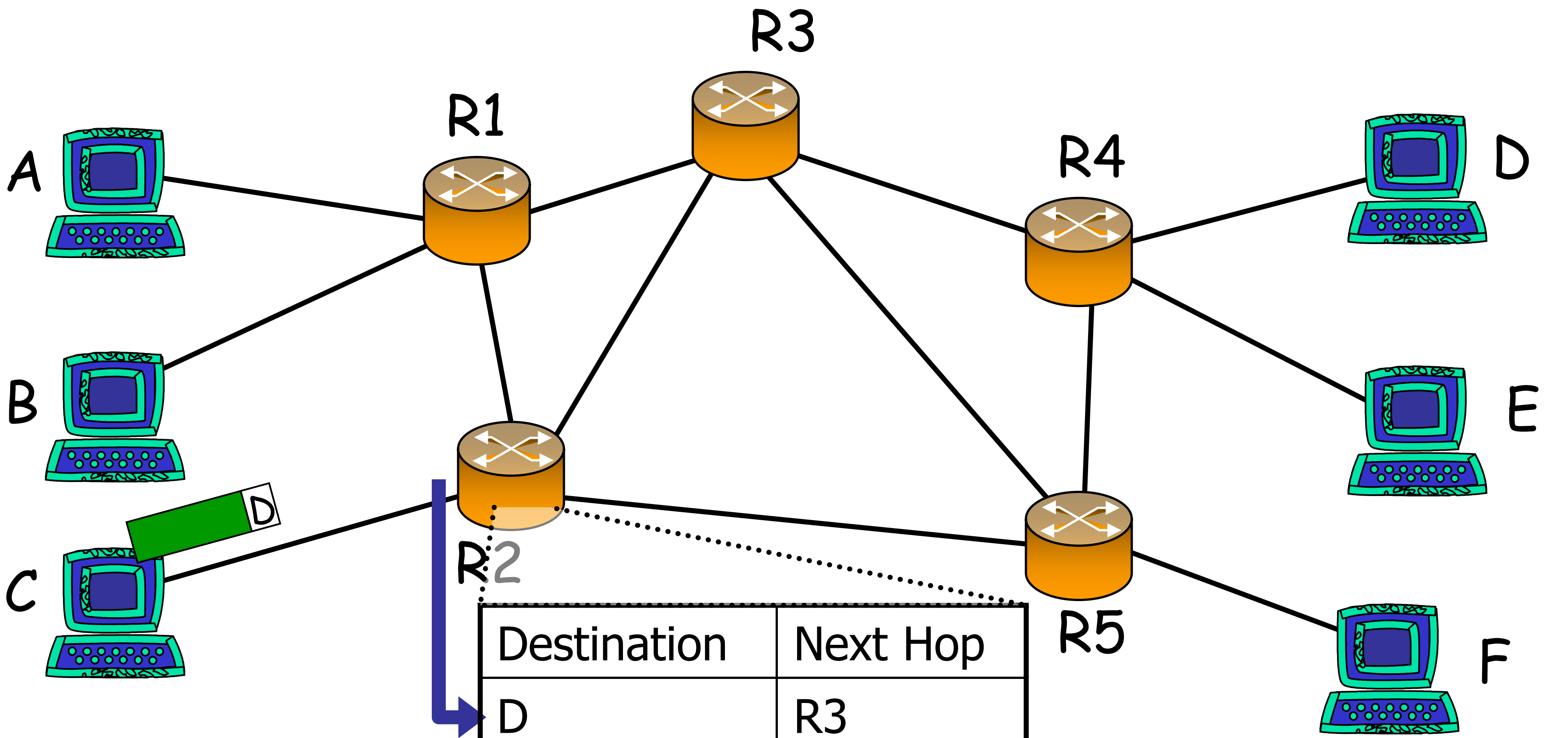
Outline

- Alternative methods for packet forwarding
- IP packet routing
- Variable prefix match

Techniques for Forwarding Packets

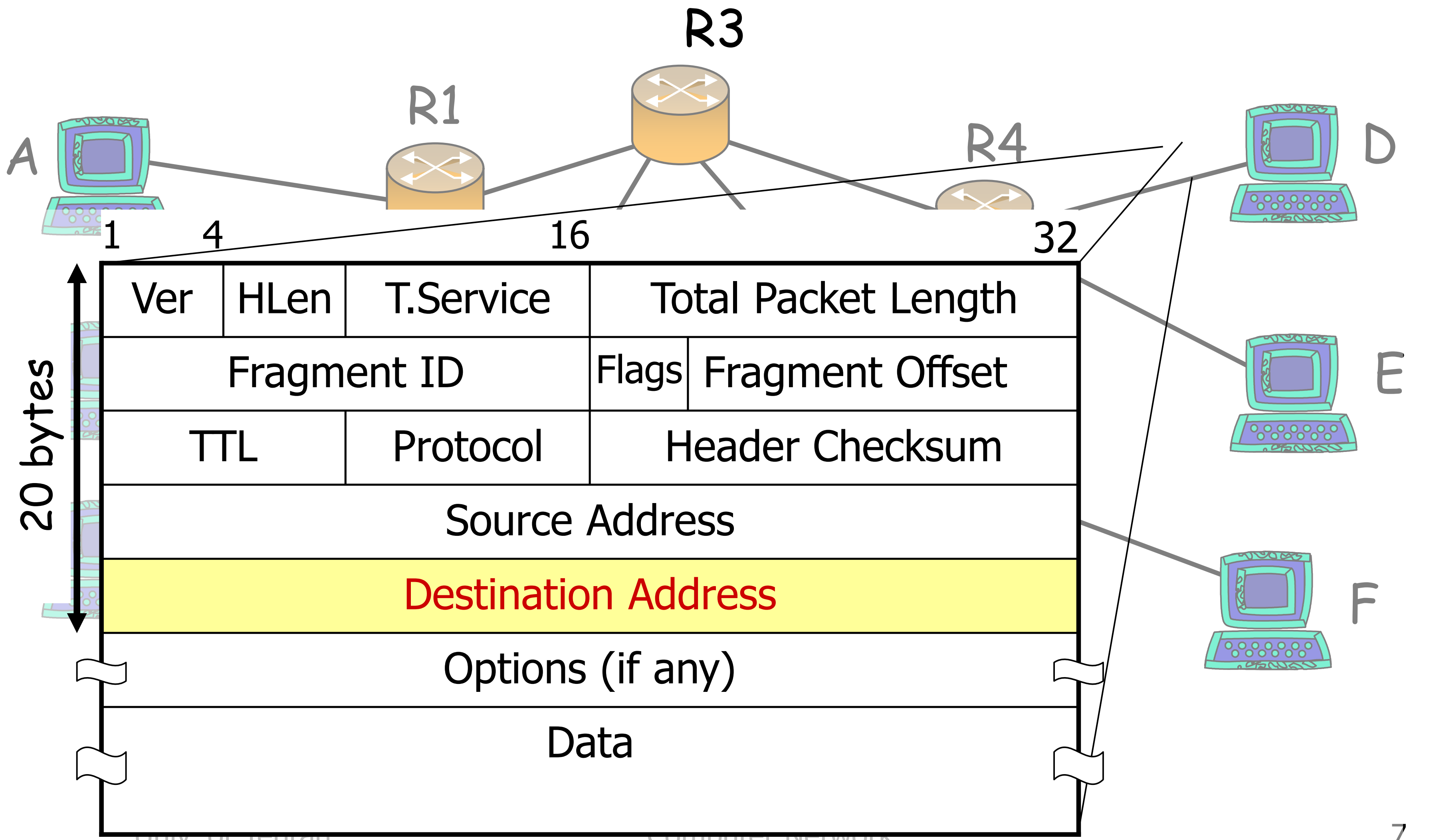
- Source routing
  - Packet carries path
- Virtual circuit switching: Table of virtual circuits
  - Connection routed through network to setup state
  - Packets forwarded using connection state
- Datagram Network: Table of global addresses (IP)
  - Routers keep next hop for destination
  - Packets carry destination address

Switching a packet



Destination	Next Hop
D	R3
E	R3
F	R5

What is Routing?



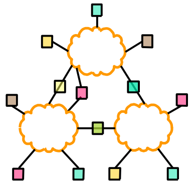
1	4		16	32
Ver	HLen	T.Service	Total Packet Length	
Fragment ID			Flags	Fragment Offset
TTL		Protocol	Header Checksum	
Source Address				
Destination Address				
Options (if any)				
Data				

Problems?

- How does a router populates the table?
  - By routing protocol
- How big is the table?
  - 3 billions entries?
- How much does it take to find a match?
  - In Ave. search half entries? 1.5 billions?
  - With 3 GHz clock & 4 cycles for each mach, it takes 2 sect

Throughput is 0.5 packet/sec, very bad!!

A better Solution



- What is the destination address?
  - An IP address
- How is the IP address?

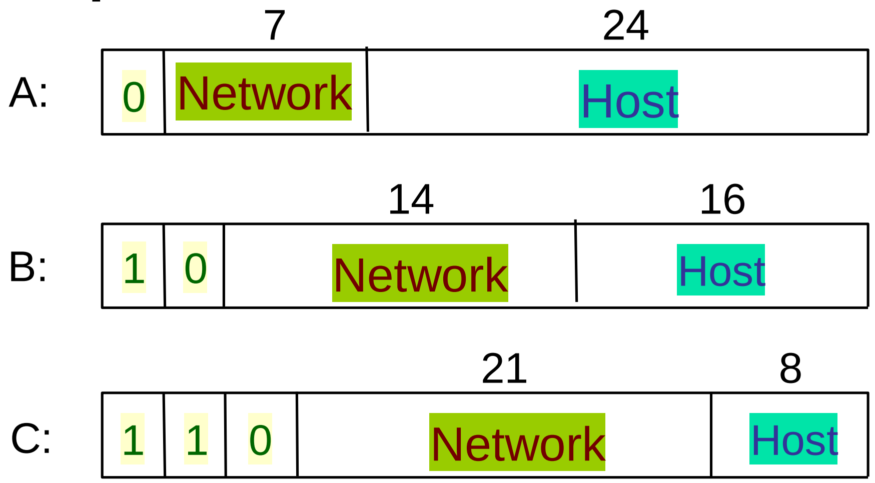
Network Address	Host Address

- **A point**: It is enough to forward packet to the network.
  - Cuts the size of the table to at least 1/100?
  - With 3 GHz clock & 4 cycles for each mach, it takes 20 msect

Throughput is 50 packet/sec, a little better

Even better Solution

- How many bits for each part in IP address?
  - Different IP classes
- Address Classes
  - Class D for Multicasting
  - Class E for experiments
- **Solution**
  - One table for each class
  - Takes network address as index to the table
  - One memory access for each maching



Then, assume 4 cycles for each memory access

Throughput is 750 millions packet/sec, Excellent!!

Outline

- Alternative methods for packet forwarding
- Variable prefix match
- Routing protocols – distance vector

IP Address Problem (1991)

- Inefficient use of Hierarchical Address Space
  - class C with 2 hosts (2/255 = 0.78% efficient)
  - class B with 256 hosts (256/65535 = 0.39% efficient)
- Address space depletion
  - In danger of running out of classes A and B
  - Class C too small for most domains
  - Very few class A – IANA (Internet Assigned Numbers Authority) very careful about giving
  - Class B – greatest problem

Solutions for IP Address

- Better utilize address space
  - Subnetting
  - Supperneting
- Locally use unofficial IP addresses (NATing)
- Use wider IP addresses
  - New version of IP, IPv6.

Supernetting

- Assign block of contiguous network numbers to nearby networks
- Called CIDR: Classless Inter-Domain Routing
- Represent blocks with a single pair

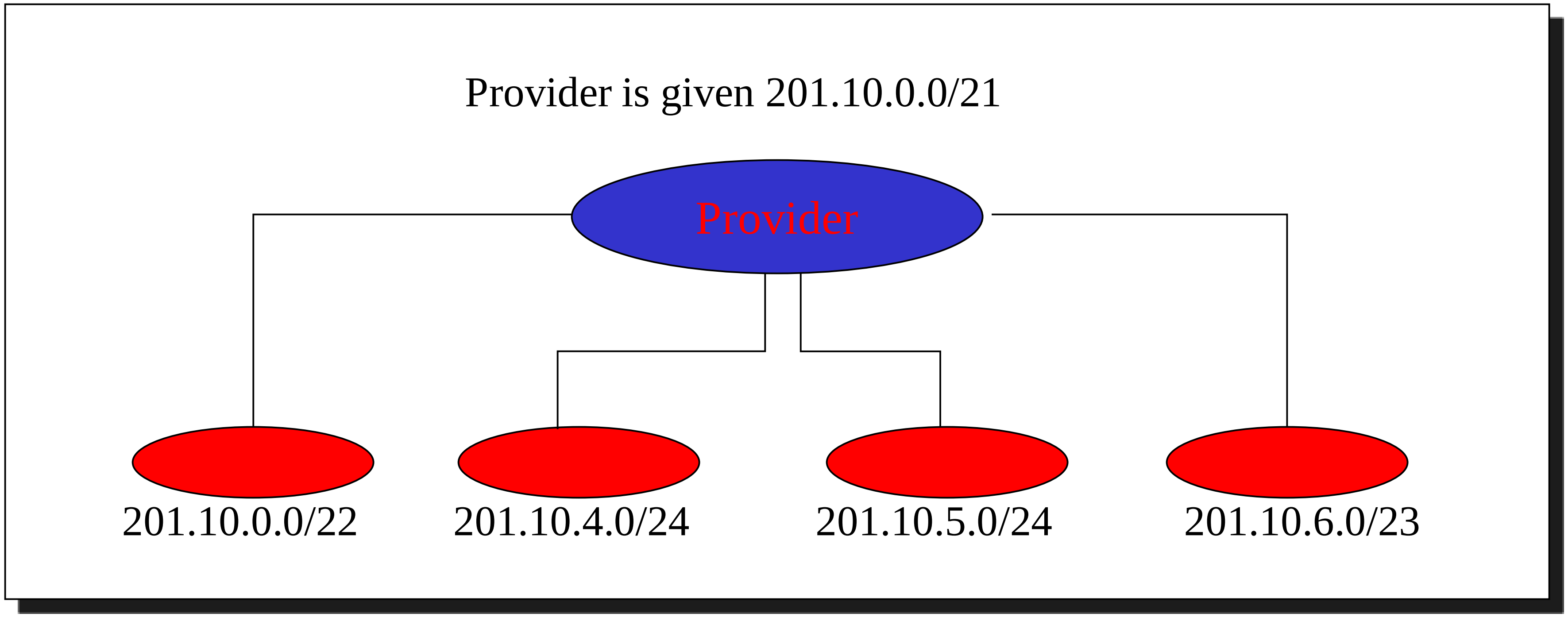
```
(first_network_address, count)
```

- Restrict block sizes to powers of 2
- Use a bit mask (CIDR mask) to identify block size
- All routers must understand CIDR addressing

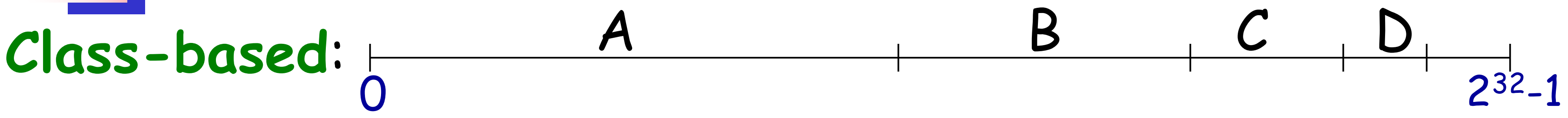
CIDR Revisited

- Supernets
  - Assign adjacent net addresses to same org
  - Classless routing (CIDR)
- How does this help routing table?
  - Combine routing table entries whenever all nodes with same prefix share same hop

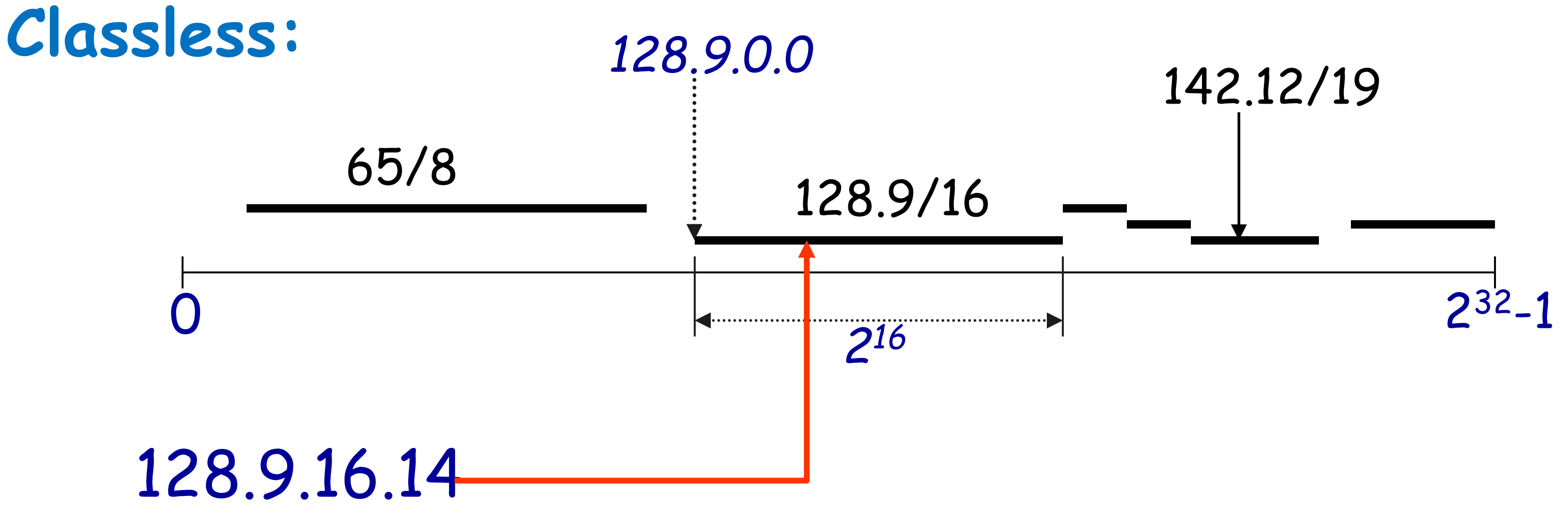
CIDR Illustration



Classless Addressing
CIDR



- Class A cover a large range of addresses

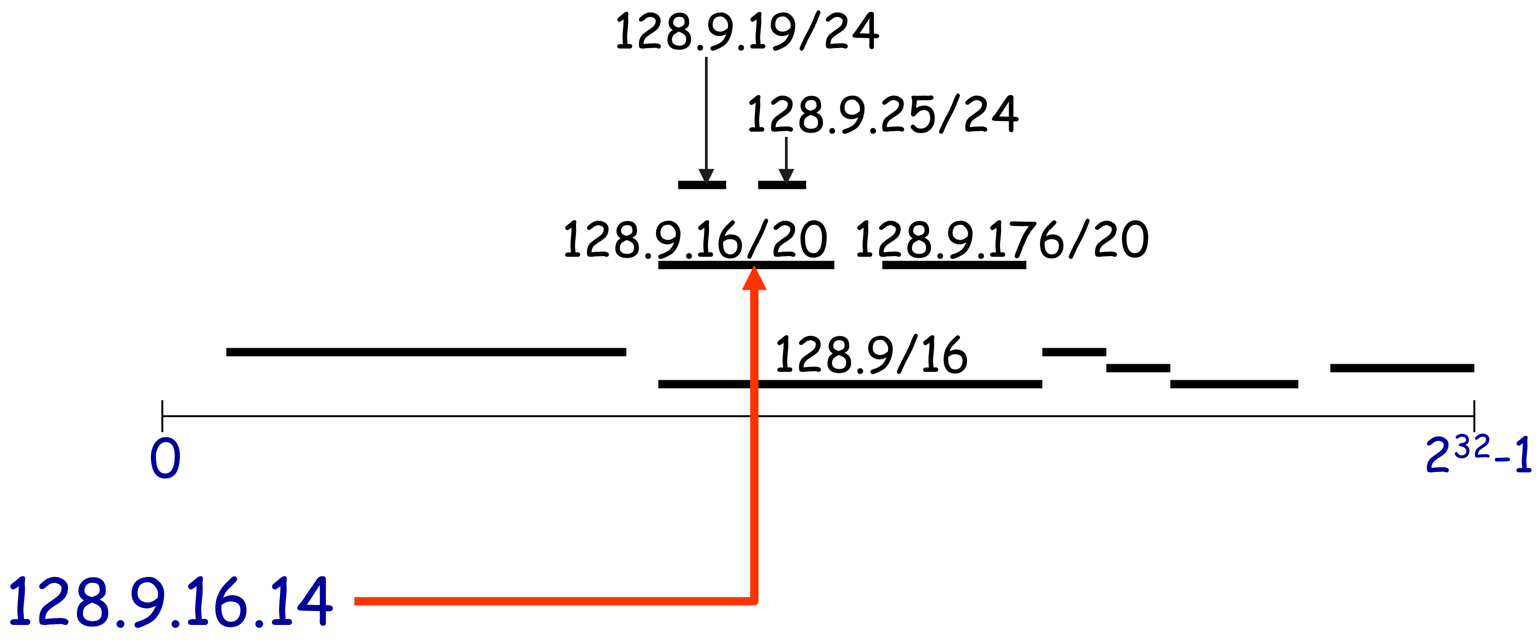


- Take necessary part of address space for network addresses.

Classless Addressing

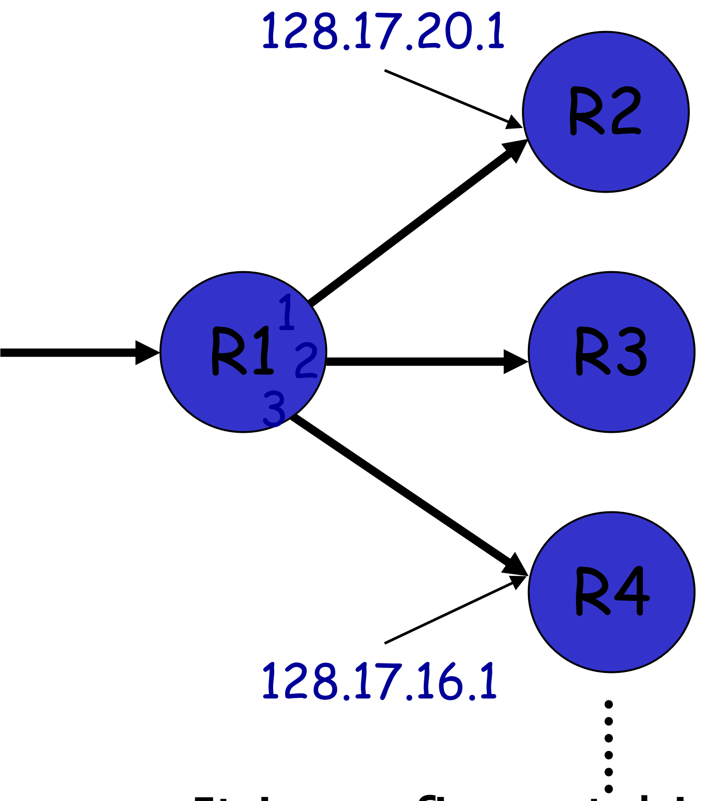
CIDR

Matching different networks in the forwarding table?



Most specific route = "longest matching prefix"

Forwarding Table



e.g. 128.9.16.14 => Port 2

Prefix	Next-hop	Port
65/8	128.17.16.1	3
128.9/16	128.17.14.1	4
128.9.16/20	**128.17.14.1**	**2**
128.9.19/24	128.17.10.1	7
128.9.25/24	128.17.14.1	2
128.9.176/20	128.17.20.1	1
142.12/19	128.17.16.1	3

- It is prefix matching in the table.

Problems?

- With CIDR we have to match all of the table not even half.
- How much does it take to find a match?
  - In Ave. 1 million entries in table
  - With 2 GHz clock & 4 cycles for each match, it takes 2 msec for each match

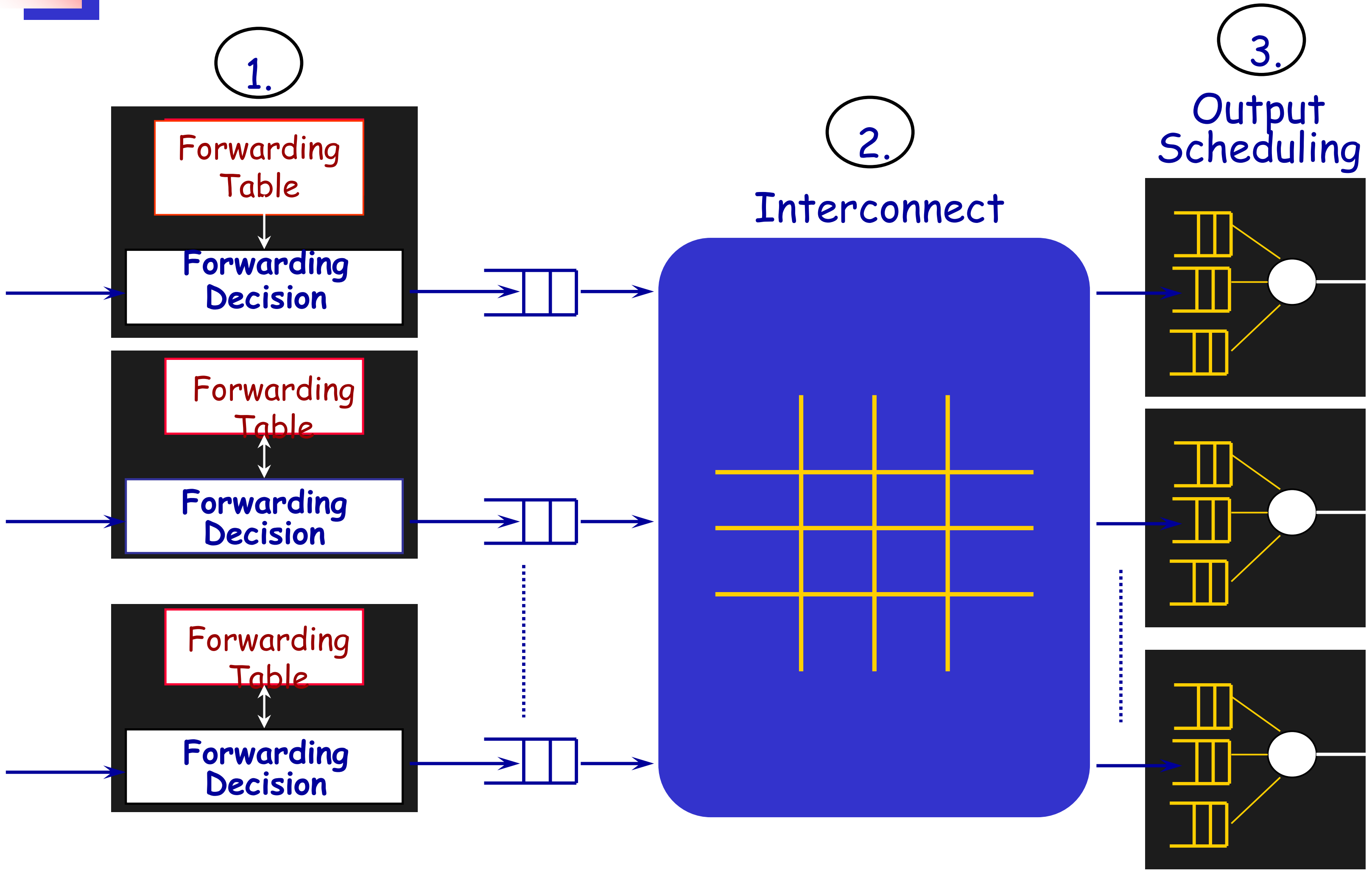
Throughput is 500 packet/sec, bad!!

Forwarding Algorithm

```
If the datagram destination address is in the local network
    deliver packet over the interface
else if the network address is in the forwarding table
    deliver packet to the corresponding next hop
else
    deliver packet to the default router.
```

- For Class address there is exact matching
- For Classless address it is prefix matching.

Inside a Router



Forwarding in an IP Router

A higher level view from router perspective.

- Lookup packet DA in forwarding table.
  - If known, forward to correct port.
  - If unknown, drop packet.
- Decrement TTL, update header Checksum.
- Forward packet to outgoing interface.
- Transmit packet onto link.

Outline

- Alternative methods for packet forwarding
- IP packet routing
- Variable prefix match

IP Lookup

- Packets are forwarded to their destination based on their destination addresses.
- Router must find the address of the next hop for each packet by finding the longest prefix mat destination addres

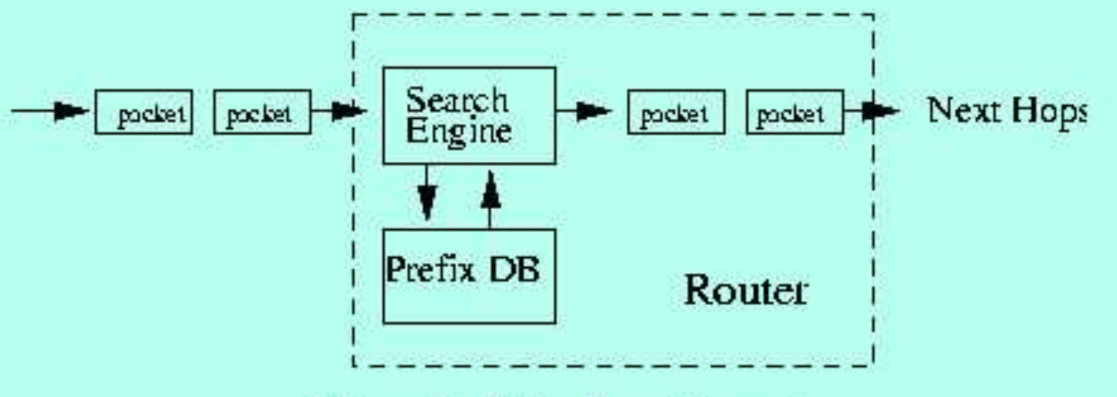
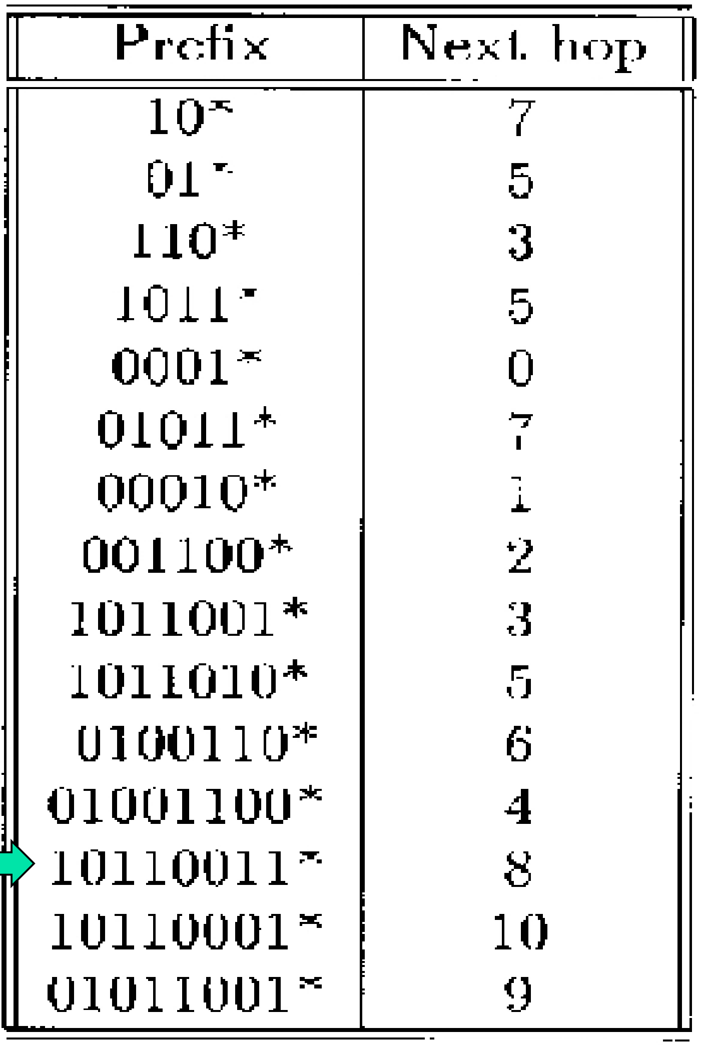


Figure 1: IP look up in routers

IP lookup

- Example:

If the a packet destination address is 101100111, the next hop will be 8 since 10110011 is the longest matching prefix.

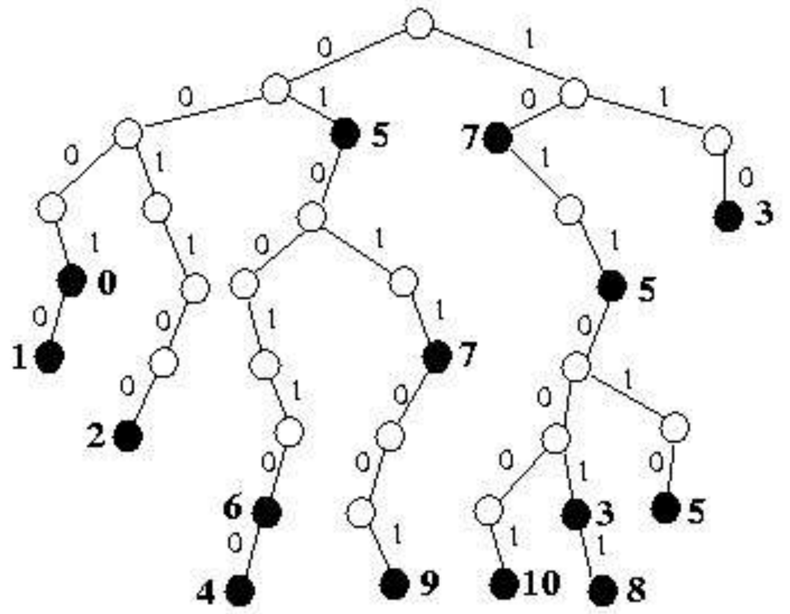


Prefix	Next hop
10*	7
01*	5
110*	3
1011*	5
0001*	0
01011*	7
00010*	1
001100*	2
1011001*	3
1011010*	5
0100110*	6
01001100*	4
10110011*	8
10110001*	10
01011001*	9

Trie Based Methods

- Trie or radix tree is a data structure in which each data element is represented by the path to the leaf and no. of internal nodes are the no. of alphabet.

•Two branching.
•Black node in the leaves represent next hops



Trie Based Methods (cont)

Trie has been the base of many methods.

There are two main problems with trie:

- The blank nodes do not correspond to any data element in the lookup table.
- The number of branching is small, 2.

Both of these add to the height of trie, waste memory and prolong the search time.

The Max search time is proportional to string length or O(32) for IPv4 and O(128) for IPv6.

Trie Based Methods (cont)

Solutions:

1. Remove or reduce blank nodes, Patricia tree, Path compression method, etc.
2. Compare 2 or more bits at each steps, Level compression and other methods.
3. Use tree or other well known data structures.
4. Other consideration: memory speed is a bottleneck!, minimize memory access.

Tree Based Lookup

- Comparing prefixes.
- Sorting prefixes
- Binary prefix Tree.
- M_way prefix tree.

Sorting prefixes

- Question? Why well-known tree structures cannot be applied to the longest prefix matching problem?
- Answer- No a well-known method for comparing prefixes.
- ***Definition***: Assume $A=a_1a_2 \dots a_n$ and $B=b_1b_2 \dots b_m$ to be prefixes of (0,1) and
  - 1. If n=m, the numerical values of A and B are compared.
  - 2. If $n \neq m$ (assume $n<m$), the two substrings $a_1a_2 \dots a_n$ and $b_1b_2 \dots b_n$ are compared. If $a_1a_2 \dots a_n$ and $b_1b_2 \dots b_n$ are equal, then, the $(n+1)^{th}$ character of string B is checked. It is considered $B>A$ if b_{n+1} is 1 and $B \leq A$ otherwise.

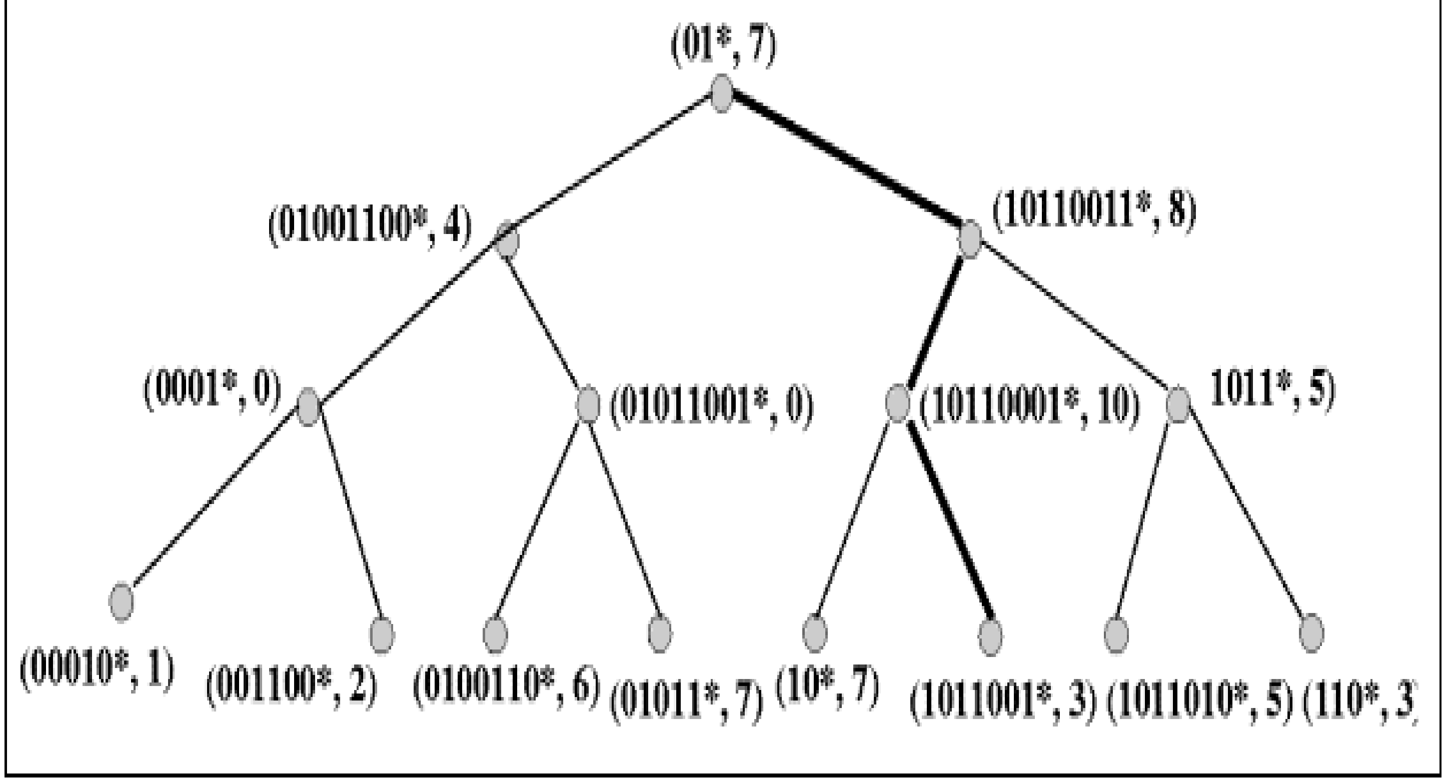
Sorting prefixes (cont)

Example-

- Sorting is a function to determine the position of each prefix.
- Prefixes of table is sorted as:

00010*,0001*,001100*,01001100*,0100110*,01011,001*,01011*,01*,10*,10110001*,1011001*,10110011*,1011010*,1011*,110*

Binary prefix tree



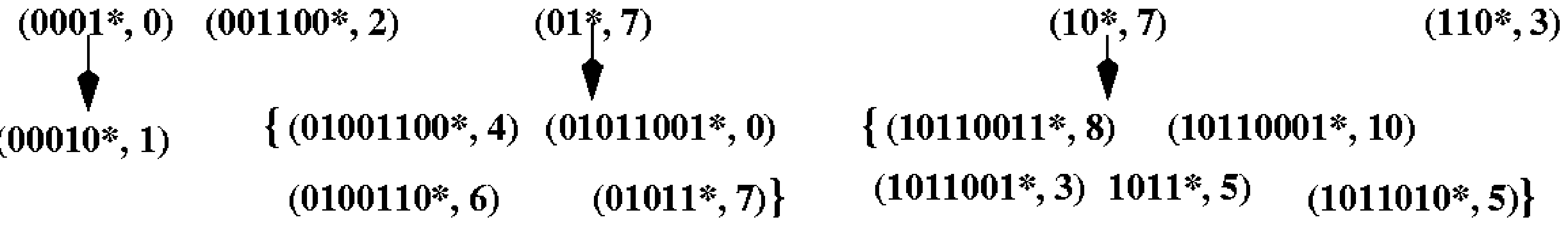
- Unfortunately, it fails for 101100001000 Why?
- Prefixes are *ranges* and not just a data point in the search space.

Binary prefix tree_(cont)

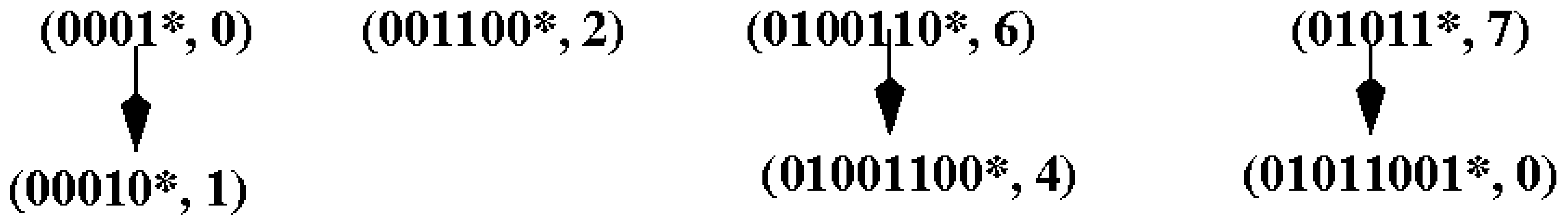
- Definition: prefixes A and B are *disjoint* if none of them is a prefix of other.
- Definition : prefix A is called *enclosure* if there exists at least one element in the set such that A is a prefix of that element.
- We modify the sort structure;
  1. *Each enclosure has a bag to put its data element on it.*
  2. *Sort remaining elements.*
  3. *Distribute the bag elements to the right and left according the sort definition.*
  4. *Apply algorithm recursively.*

Binary prefix tree (cont)

- Example- Prefixes in table 1. First step.



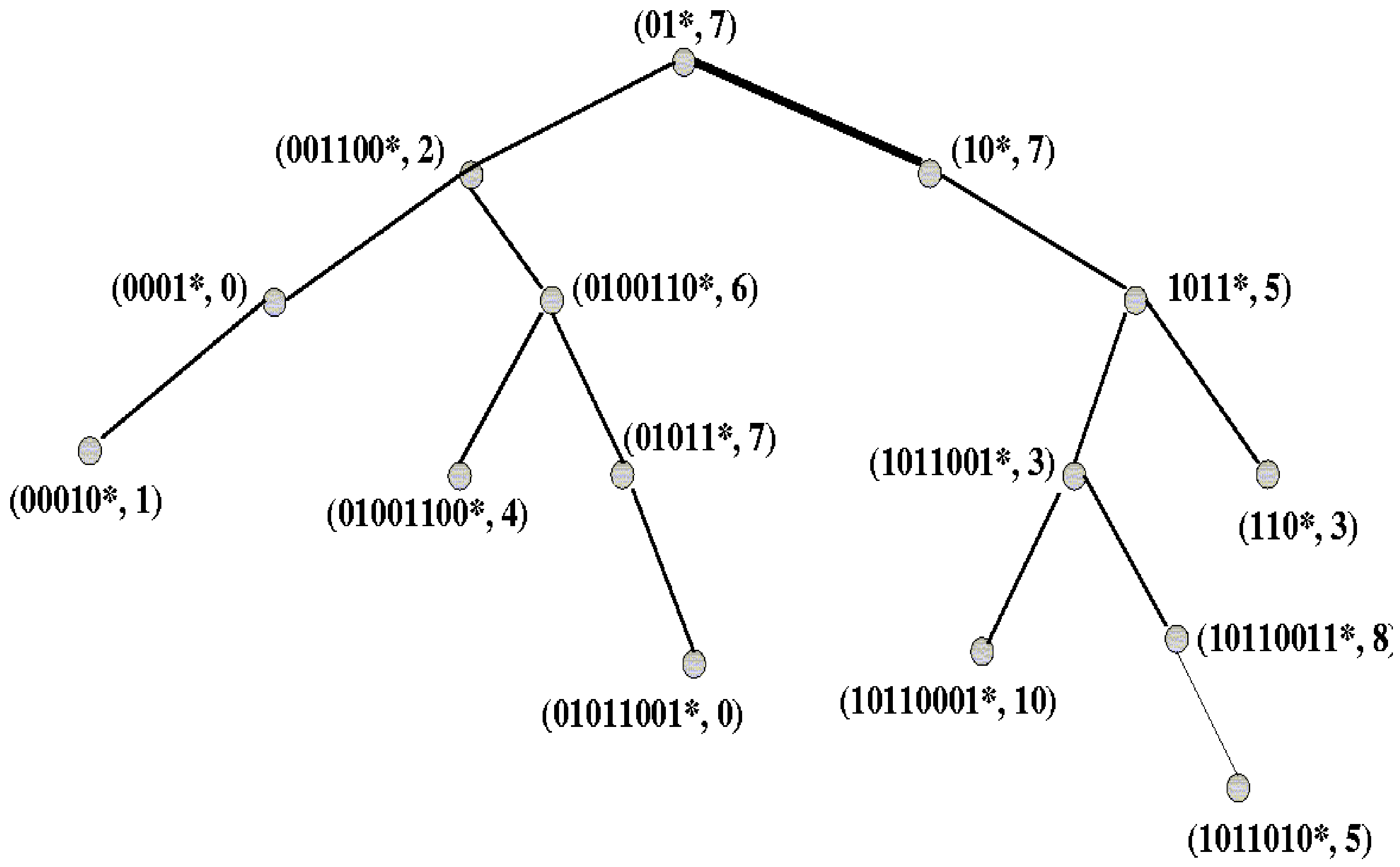
The second step,



Note- enclosures are in the higher level than the contained elements. (important!)

Binary prefix tree (cont)

- The final tree structure



M_way prefix tree

- **Problems** with the binary prefix tree.
  - Two way branching.
  - The structure is not dynamic and insertion may cause problems!.

1. Divide by ***m*** after sorting the strings
   - Static m_way tree.
2. Build a dynamic data structure like B-tree.
   - How to guarantee enclosure to be in the higher level than its contained elements.
   - Define node splitting and insertion.

M_way prefix tree *(Cont)*

- Node splitting: Finding the split point.
  1. Take the median if the data elements are disjoint.
  2. If there is an *enclosure* containing other elements, take it as split point.
  3. Otherwise, take an element which gives the best splitting result.
- Note, this does not guarantee the final tree will be balanced.

M_way prefix tree *(Cont)*

- Insertion:
  1. If the new element is not an enclosure of any prefix in the tree, find its place and insert it in the corresponding leaf, like B-tree.
  2. Otherwise, replace it with its shortest enclosed element.
  3. Reinsert the replace element and all elements in the substrees . It may cause space division.
- Building tree is similar to building B-tree.

An Example

Routing Table

Prefix	Abbrv.	Prefix	Abbrv.
10	-	1101110010	K
01	-	10001101	L
110	-	11101101	M
1011	-	01010110	N
0001	-	00100101	O
01011	-	100110100	P
00010	-	101011011	Q
001100	A	11101110	R
1011001	B	10110111	S
1011010	C	011010	T
0100110	D	011011	U
01001100	E	011101	V
10110011	F	0110010	W
10110001	G	101101000	X
01011001	H	101101110	Y
001011	I	00011101	Z
00111010	J	011110110	II

M-way prefix tree (cont)

- We insert prefixes randomly.
- The tree uses 5 branching factor (at most 4 prefixes in each node)
- Insert 01011, 1011010, 10110001 and 0100110. Then, adding 110 cause overflow. Split node

↙ 10110001 ↘

(0100110,01011)   (1011010, 110)

(all element are disjoint)

M-way prefix tree (cont)

- Insert 10110011, 1101110010, 00010. Adding 1011001 causes overflow.

↙ 10110001 ↓ 1011010 ↘

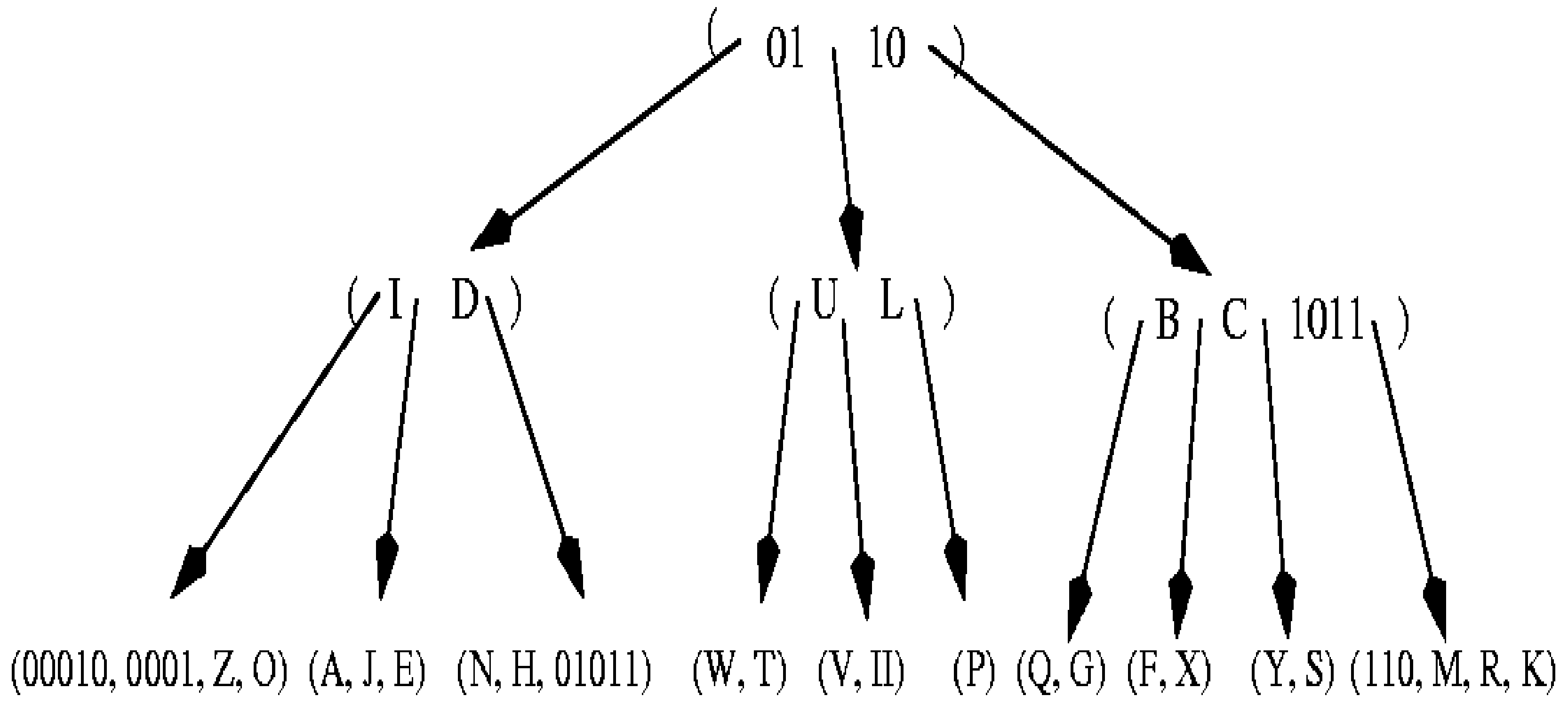
(00010,0100110,01011) (1011001,10110011) (110,1101110010)

(case 3 of splitting)

- Latter adding 1011 cause problem. It is the case of adding an enclosure. We will have space division.

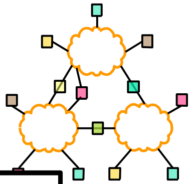
M-way prefix tree (cont)

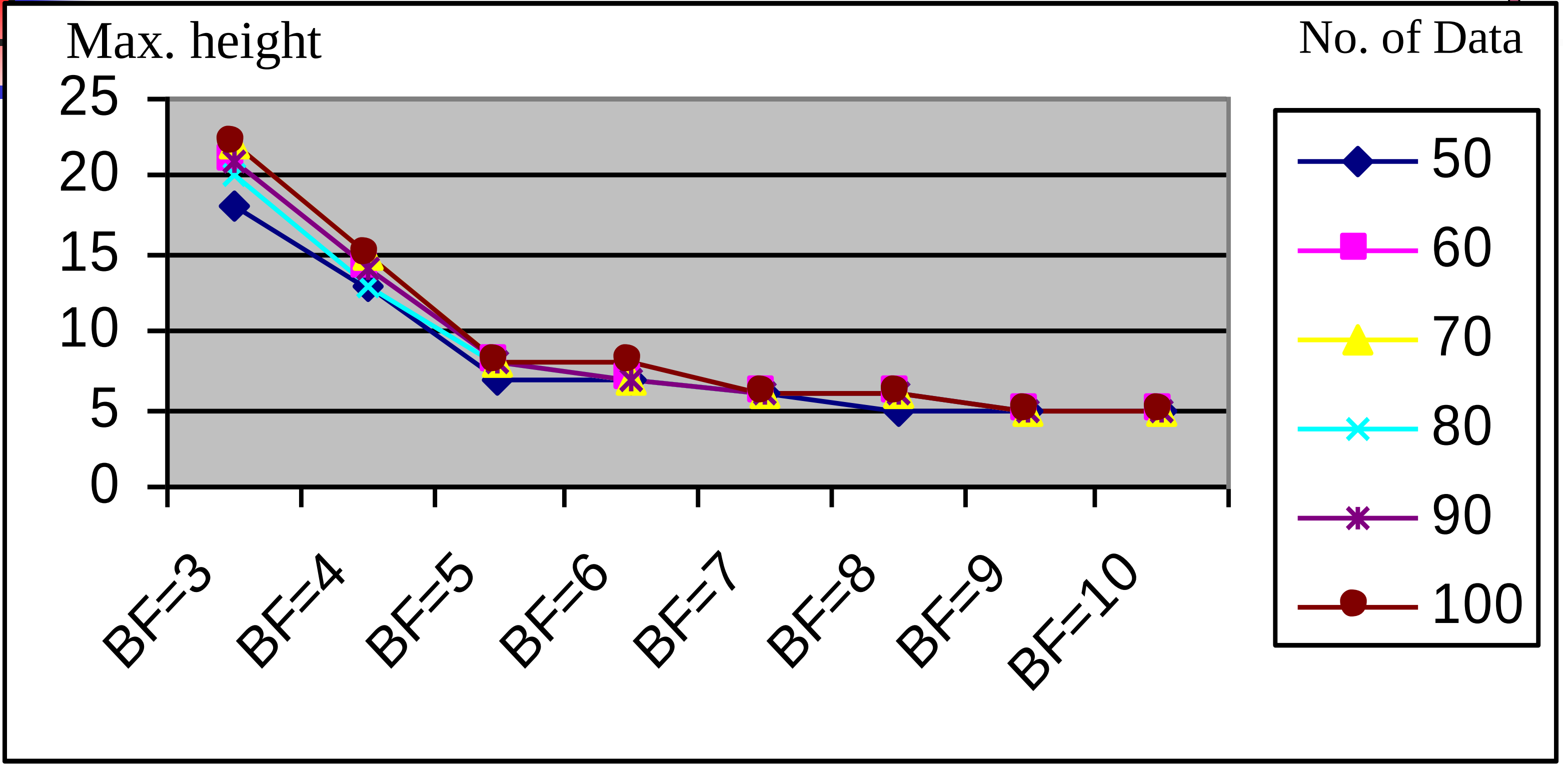
- The final tree



• The tree supersede B-tree or B-tree is a special case of this tree. Then, when data element are relatively disjoint, the height of tree is $\log_M N$.

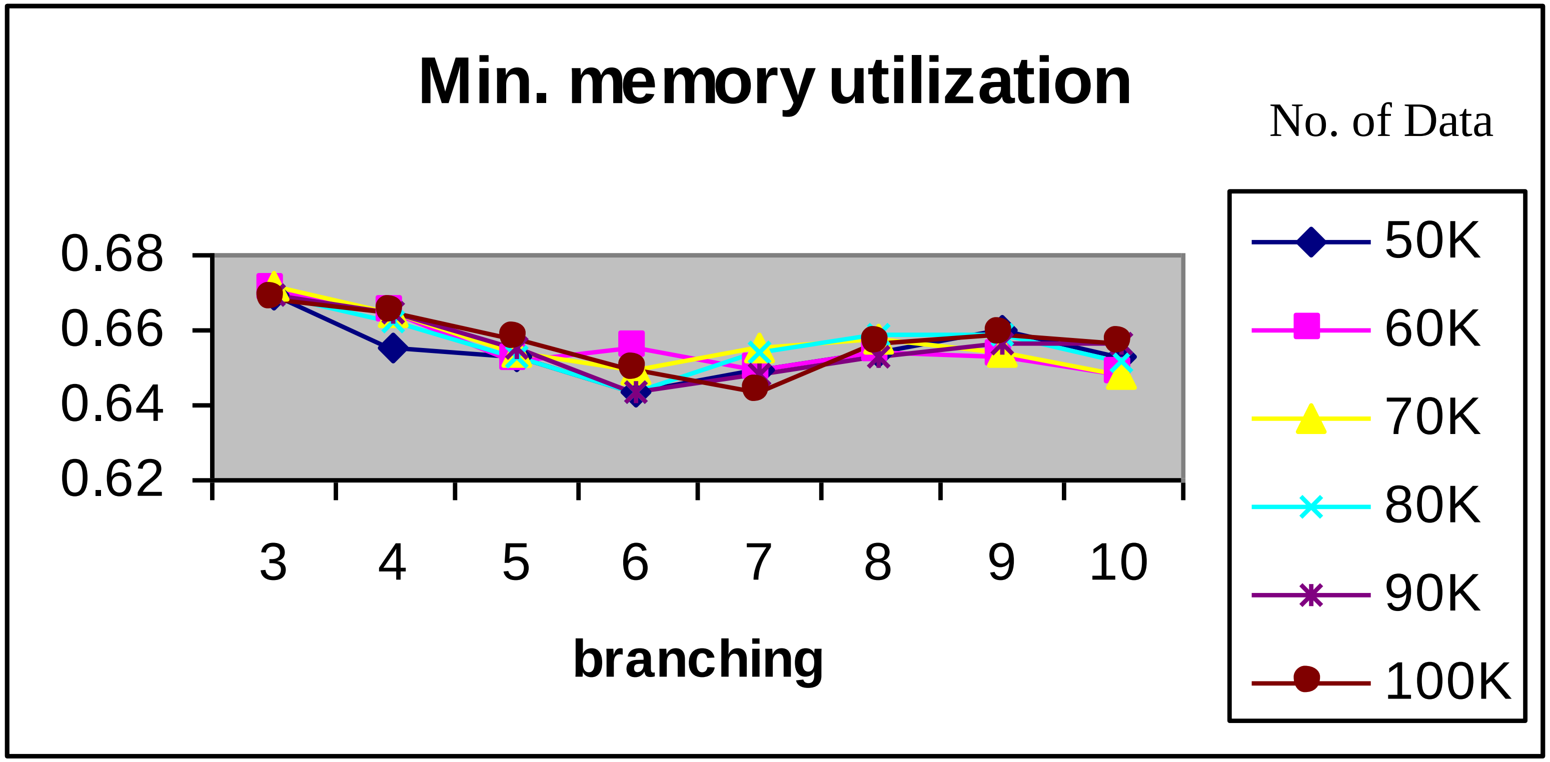
DMP-Tree





- BF is Branching factor in the internal nodes.
- No. of Data is in1000s.

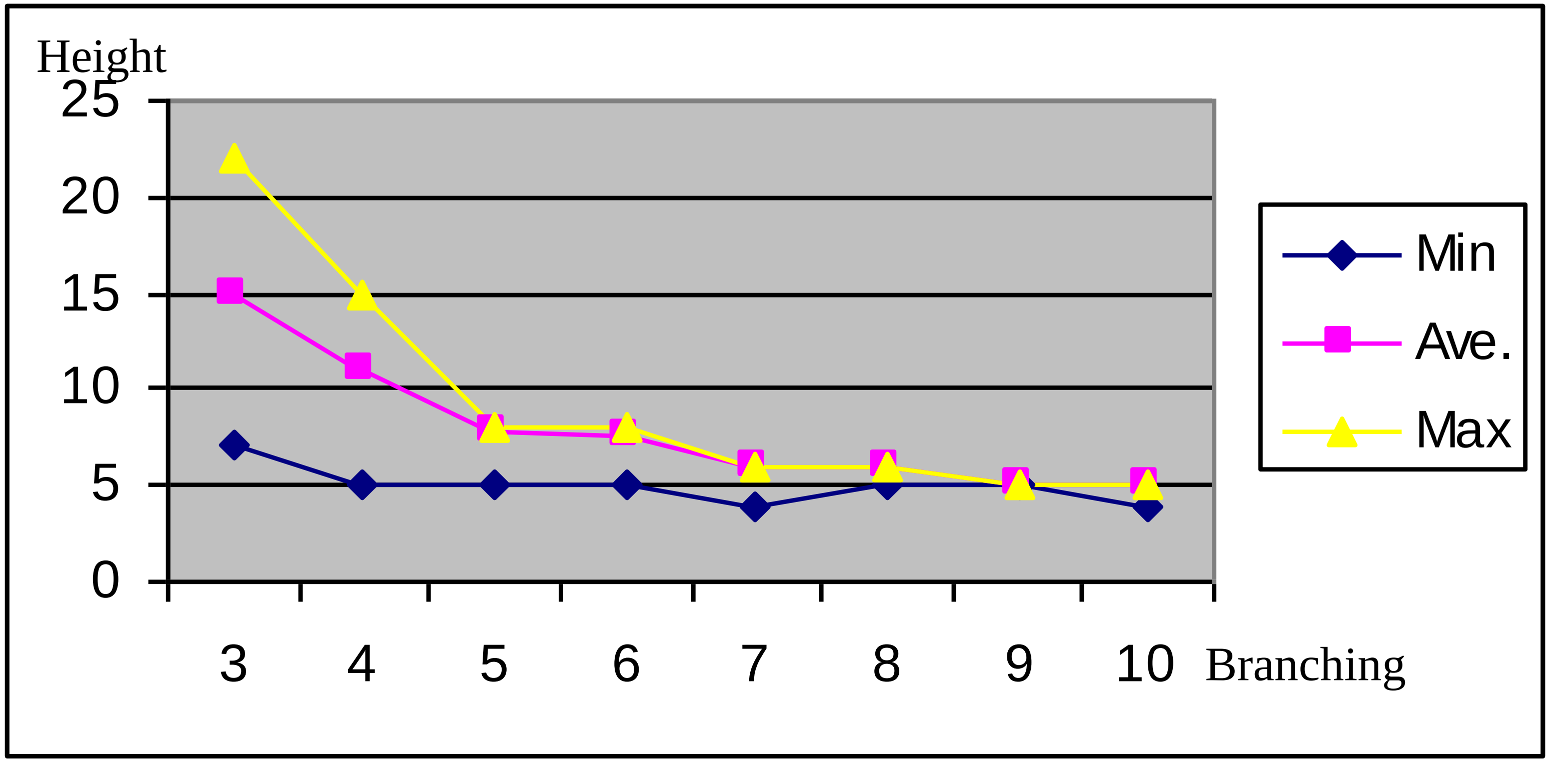
DMP-Tree



- Number of prefixes are in the right.

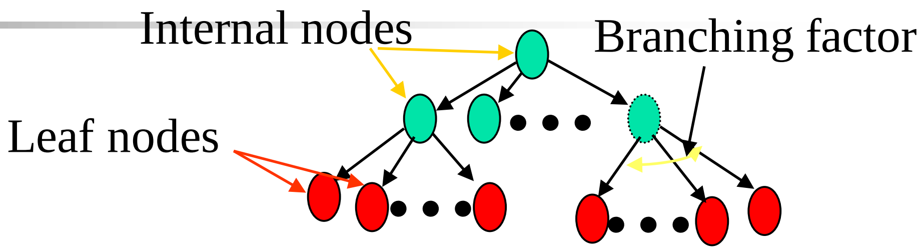
DMP-Tree

- Height of tree for 100K data prefixes.



Implementation: Tree Nodes

- Internal nodes.



Addr 1 [?]	Prefix 1 [33]	Port [?]	Addr 2 [?]	Prefix 2 [33]	port [?]	…	Addr N+1

- Each prefix has a left and right pointer which are pointing to left and right subtrees respectively.
- We can have N prefixes in each internal node. Then, N+1 is the branching factor.
- The bigger N, the faster search time, but the more logic is needed.
- Port is the address of the port in the switch to which the packet will be sent.

Tree Nodes

- Leaf nodes.

Prefix 1 [33]	port [?]	Prefix 2 [33]	port [?]	…

- There is no left and right subtree pointers.
- The number of prefixes in the leaf node is M.
- The leaf nodes are stored in a off chip memory to make the scheme scalable to the large number of prefixes.

Branching Factor

- What is the best number for N? (Branching factor)
  - The bigger N, the faster search process. (Fact 1)
  - The bigger N, the more memory pins are and usually the more memory Bandwidth is needed (Fact 2).
  - The bigger N, the more logic we need to process the node (Fact 3).
- Simulation result shows
  - The bigger N, the better memory utilization in the memory.
  - For N $\geq$ 8, the max. height of the tree does not decrease considerably.

Simulation result

Total memory: assuming one memory block and OC-192.

# of Prefixes	required Mem.	Branching Factor	Mem. Pins	Mem BW (G/s) max	Max mem Access	Mem. Size (on chip)mm^2	Max heights
64K	5.4 Mbits	15	897	89.7	4	81	5
64K	5.5 Mbits	11	655	65.5	4	82	5
64K	5.4Mbit	9	527	52.7	4	81	5
64K	6 Mbit	6	335	46.9	6	96	7
64K	6.6 Mbit	5	275	44	7	110	8
64K	6.5 Mbit	4	207	62.1	14	112	15
100K	8.3 Mbit	15	897	89.7	4	122	5
100K	8.5 Mbit	11	655	65.5	4	125	5
100K	8.3	9	527	63.24	5	122	6
100K	9 Mbit	6	335	53.6	7	135	8
100K	9.1Mbit	5	275	49.5	8	140	9
100K	9.5	4	207	62.1	14	150	15
30K	2.6	9	527	52.7	4 expected	40	5 expected

Branching Factor

- It seems any number between 8-16 is reasonable. But, N=9 gives a better search time and memory size.
- Assuming 9 branching factors in the internal node, %50 node utilization and 128K prefixes, we need max. 128K/4.5= 28.5K address. Then, 15 bit address for left and right pointers are more than enough. But, we need more for off chip addressing

Branching Factor

- In order to make the internal node branching and leaf node branching even, M=10.
- If we want to read a node at once, we will need 41x10=410 pins which is difficult to support in one chip.
- We can divide a node in two and read/write in two clock cycles. This reduce the memory pins to 205 which is affordable.

Memory requirement

- Prefix tree: 128K prefixes, N = 9 (BF) and M=10 (BF in leaves), %80 of prefixes will be in leaves, assume %65 node utilization,

  # of ave prefixes in a leaf node node = 10*0.6 5= 6.5

  # of leaf nodes ≅ 128Kx%80x2/6.5 = 31.5K and %10 overhead ≅ 35 K

  Total off chip memory = 35K x 205(Mem BW) = 7.2 Mbits

  We need 16 bits for addressing. 1 bit for internal/external.

  # of internal nodes= 128Kx%20/5.8=4.41K and %10 overhead ≅4.9 K

  Total on chip memory=4.9Kx529 ≅ 2.6Mbits

- Port to link address mapping table.

  For each port corresponding link address

  Max. 256 ports, on chip, some mem for indexing

Link Addr
48 bit addr.

Memory requirement

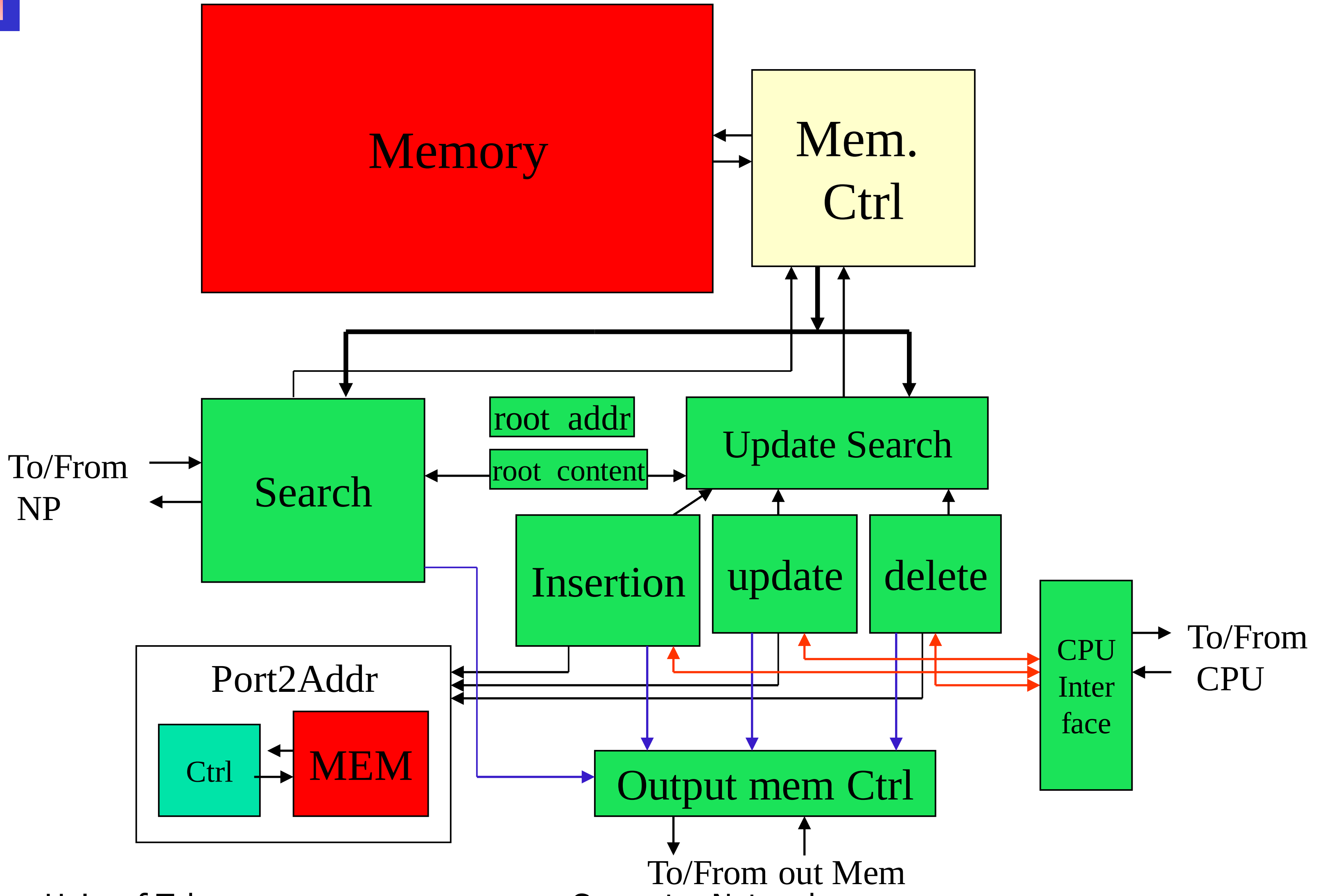
In summary:

# of Prefixes	on chip memory Mbits	Branching Factor	On chip Mem. Pins	Off chip memory Mbits	mem Access(search)	Mem. Size (on chip) mm^2	Off chip mem pins
128K	2.6	10	529	7.2	5	40	250

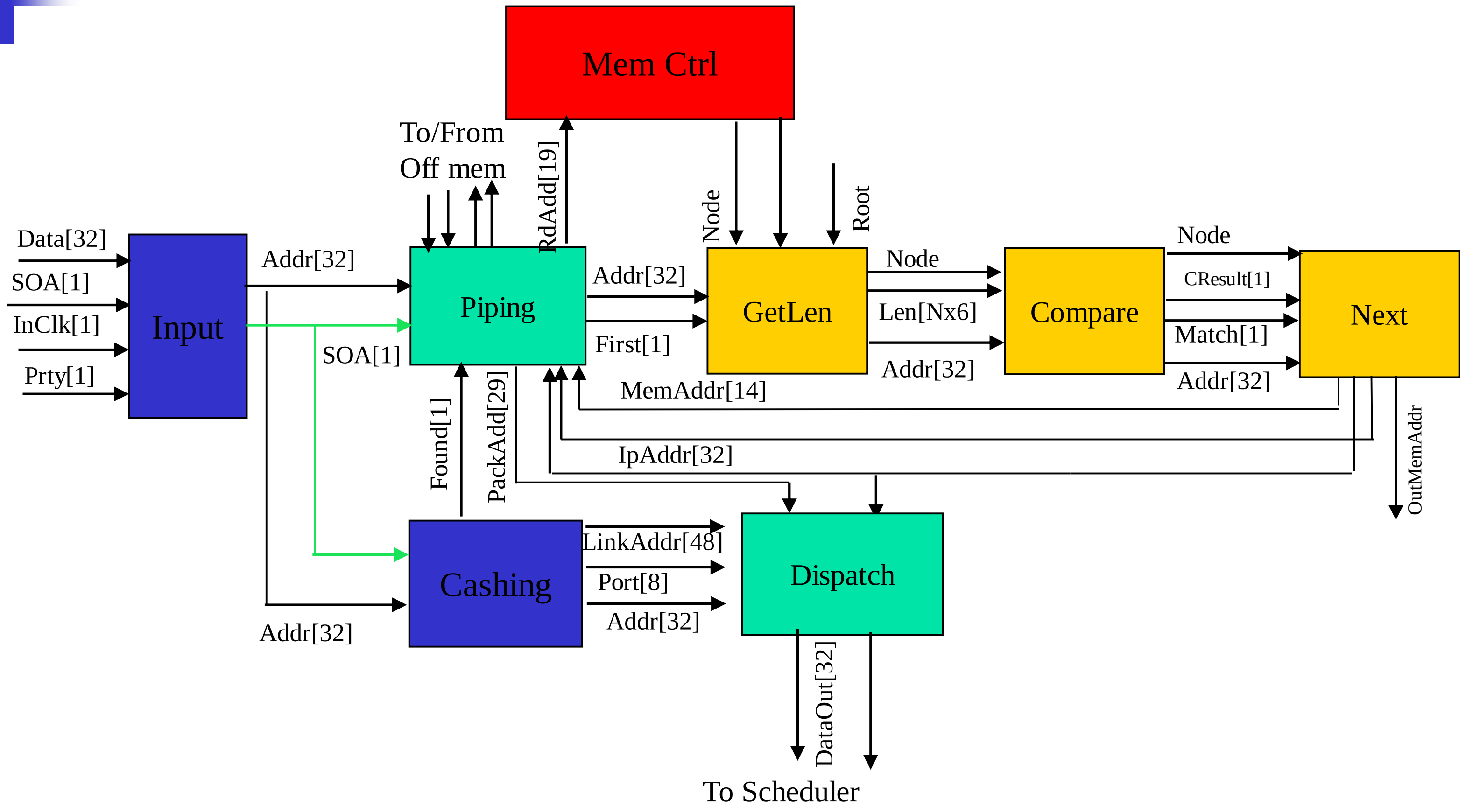
Note:

- Branching factor is the # of branching in internal nodes.
- The size of the memory scales with the size of data or # of prefixes.
- Power dissip. depends on the r/w freq, current & core voltage
- Considering Faraday Mem. Modules
  A 10Kx32 bits single port mem size is 36x1.45 mm^2.

Overall Design



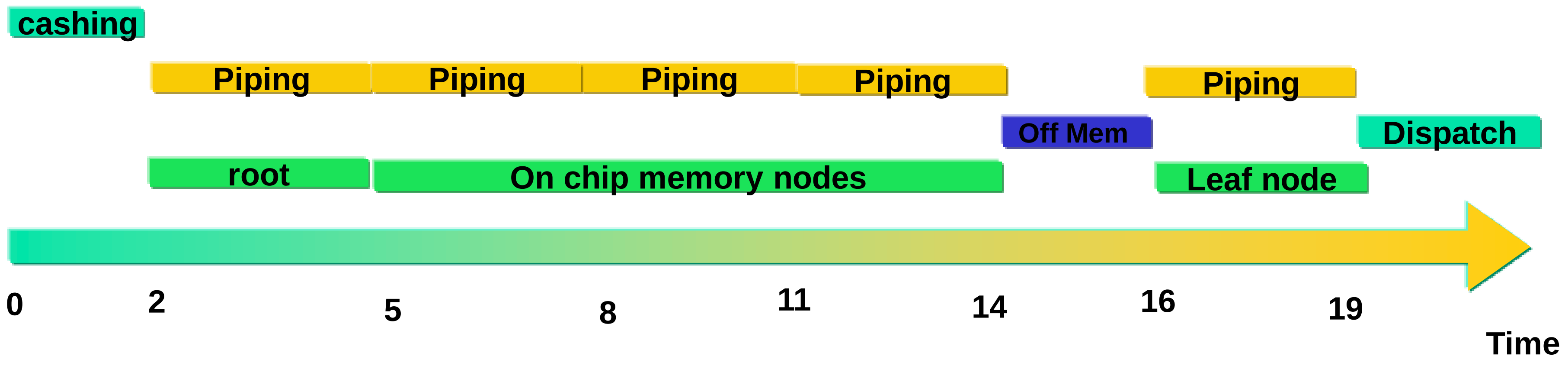
Search Path



There are data assertion signals between blocks which has not been shown every where because of space limitation.

Search Process

operations: This is for large prefixes (50K &up)



- This operation is for an IP address lookup
- Piping is the bottleneck in the system and in ave. take 5 cycles.
- Assuming 100 MHZ operation

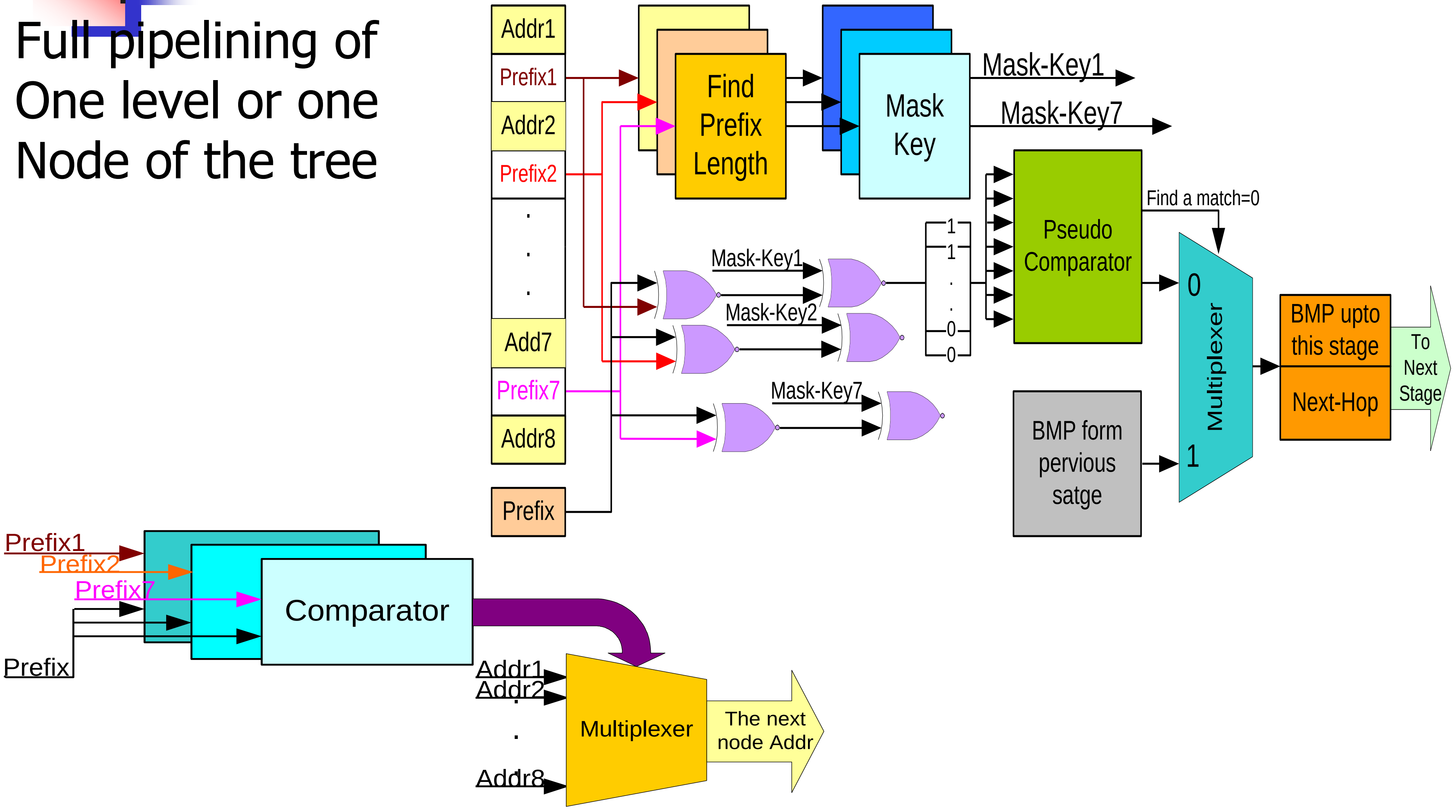
  # of packets = $10^9/50$ = 20 Million

  Line speed = 512x20 = 10.24 G for 64 byte packets.

  = 256x8x20= 41 G for 256 byte packets.
- It is possible to support higher speeds with duplicating pipe.

Full Pipelining of node

Full pipelining of One level or one Node of the tree



Full Pipelining of whole tree

- ✓ Whole tree can be pipelined by repeating the logic with the number of tree level.
- ✓ Each level of tree needs to be stored in a different memory module.
- ✓ In average, a lookup can be done in one clock cycle.

Software Implementation

- How to represent prefixes
  - With adding 1 in the end and filling with 0's
    - Needs around 3 clk cycles to extract length, for two 6 cycles.
- How to compare
  - According to definition
    - Check which one's length is bigger (1 clock)
    - Makes lengths equal (3-4 clock)
    - Compare prefixes (1 clock)
    - Check next bit if they are equal (3 clock)
  - at least 15 clock to compare.
- With 9 branching 6x9x15=810 cycles plus overhead and memory access. Over 1000 cycles.

Software Impl: improvement

- We know which one’s length is bigger.
- We know the length of one prefix, IP addr.
- No need to compare all fields, in ave, 6 compare (5-6 clock)
- How to compare
  - According to definition
    - Extract prefix length (3-4 clock)
    - Makes lengths equal (3-4 clock)
    - Compare prefixes (1 clock)
    - Check next bit if they are equal (3 clock)
  - at least 10 clock to compare.
- With 9 branching 6x6x10=360 cycles plus overhead and memory access.

Software Impl: HOSM format

- ✓ Prefix can be represented by adding a zero to its tail, and then filling it with ones to make its length equal to the biggest possible prefix length plus one, here 32. (e.g.: 101* represents as 101*0*11….1.)
- ✓ Numerical comparison between prefixes in *Hosm-Format* is equivalent to prefix-comparison of the original Definition.
- ✓ We can compare prefixes in one cycle.
- ✓ With 9 branching 6x6=36! cycles plus overhead and memory access.

New Service requirements

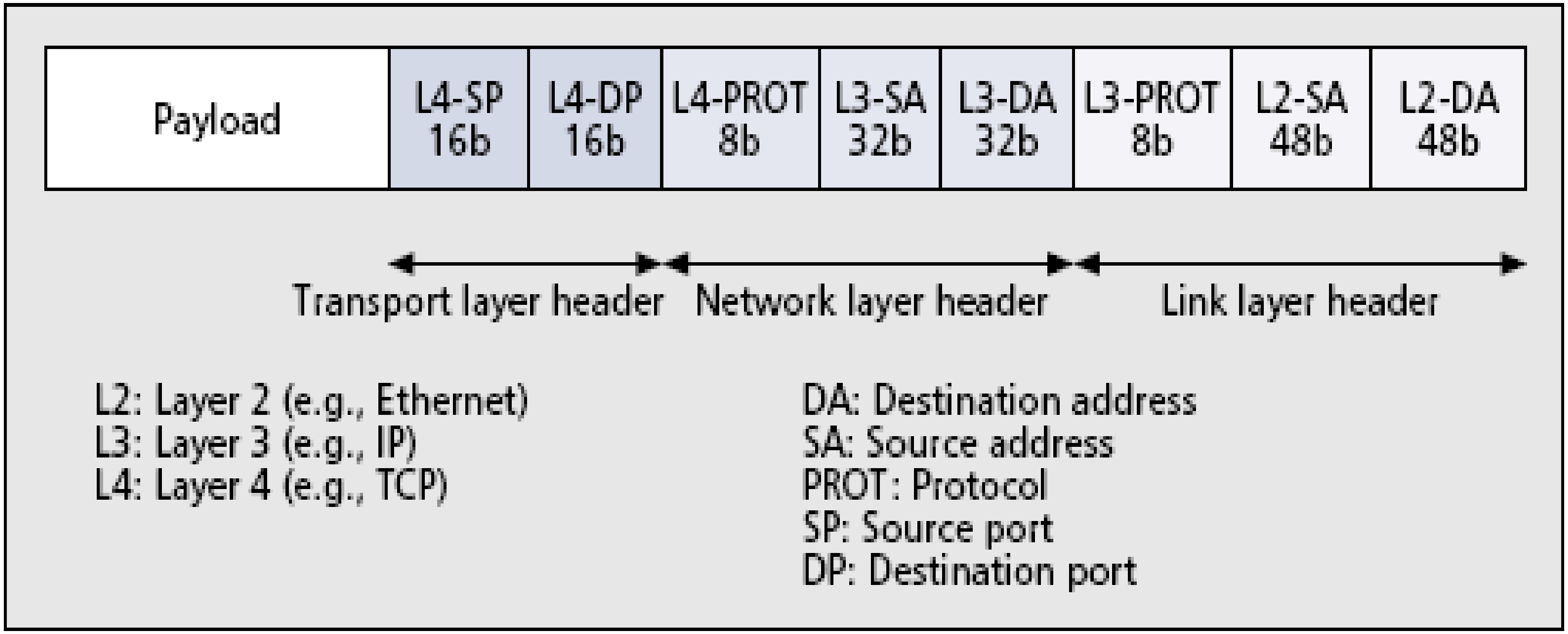
- Blocking traffic sent by insecure sites (firewalls)
- Preferential treatment for premium traffic (resource reservation)
- Routing based on traffic type and source (QoS routing)
- Accounting for each user (Billing)
- Limiting rate from some sources (Shaping)

We need to distinguish traffic based on destination, source, application type, etc or to **Classify packets**.

Packet Classification?

- Informally
  - Identifies the flow a packet belongs to, based on one or more fields in the packet header
  - The ability to match each packet against a database of rules
  - The process of categorizing packets into “flows” in an Internet router

Packet Header



- Some of the header fields be used for classifying the packet. Although not shown, *higher-layer (application level)* headers may also used.

Example Classifier

Rule	Network-layer Destination (address/ mask)	Network-layer Source (address/mask)	Transport-layer Destination	Transport-layer Protocol	Action
R1	152.163.190.69/ 255.255.255.255	152.163.80.11/ 255.255.255.255	*	*	Deny
R2	152.168.3.0/ 255.255.255.0	152.163.200.157/ 255.255.255.255	eq www	udp	Deny
R5	152.163.198.4/ 255.255.255.255	152.163.160.0/ 255.255.252.0	gt 1023	tcp	Permit
R6	0.0.0.0/0.0.0.0	0.0.0.0/0.0.0.0	*	*	Permit

Packet Header	Network-layer Destination	Network-layer Source	Transport-layer Destination	Transport-layer Protocol	Best matching rule, Action
P1	152.163.190.69	152.163.80.11	www	tcp	R1, Deny
P2	152.168.3.21	152.163.200.157	www	udp	R2, Deny
P3	152.168.198.4	152.163.160.10	1024	tcp	R5, Permit

Performance Metrics

- Search speed
- Storage requirement
- Scalability in classifier size
- Scalability in the number of header fields
- Update time
- Flexibility in specification

Next Lecture: Routing

- Interdomain routing.
- References
  - Hari Balakrishnan notes about Interdomain from Mit
  - RFC1771-2-3-4., rfc1267, rfc4632 : Main BGP RFCs.
  - RFC1965: AS confederations for BGP
  - Craig Labovitz and .. "Delayed internet Routing Convergence"
  - John Stewart III: "BGP4 - Inter-domain routing in the Internet"
  - HLP: A Next Generation Interdomain Routing Protocol
  - "Some Foundational Problems in Interdomain Routing"