Design of High Performance Internet Routers/Switch And Openflow Switch Packet Classification (高效能網際網路路由器/交換器之設計 Openflow交換器之封包分類)

張燕光

資訊工程學系

Dept. of Computer Science & Information Engineering,

國立成功大學

National Cheng Kung University

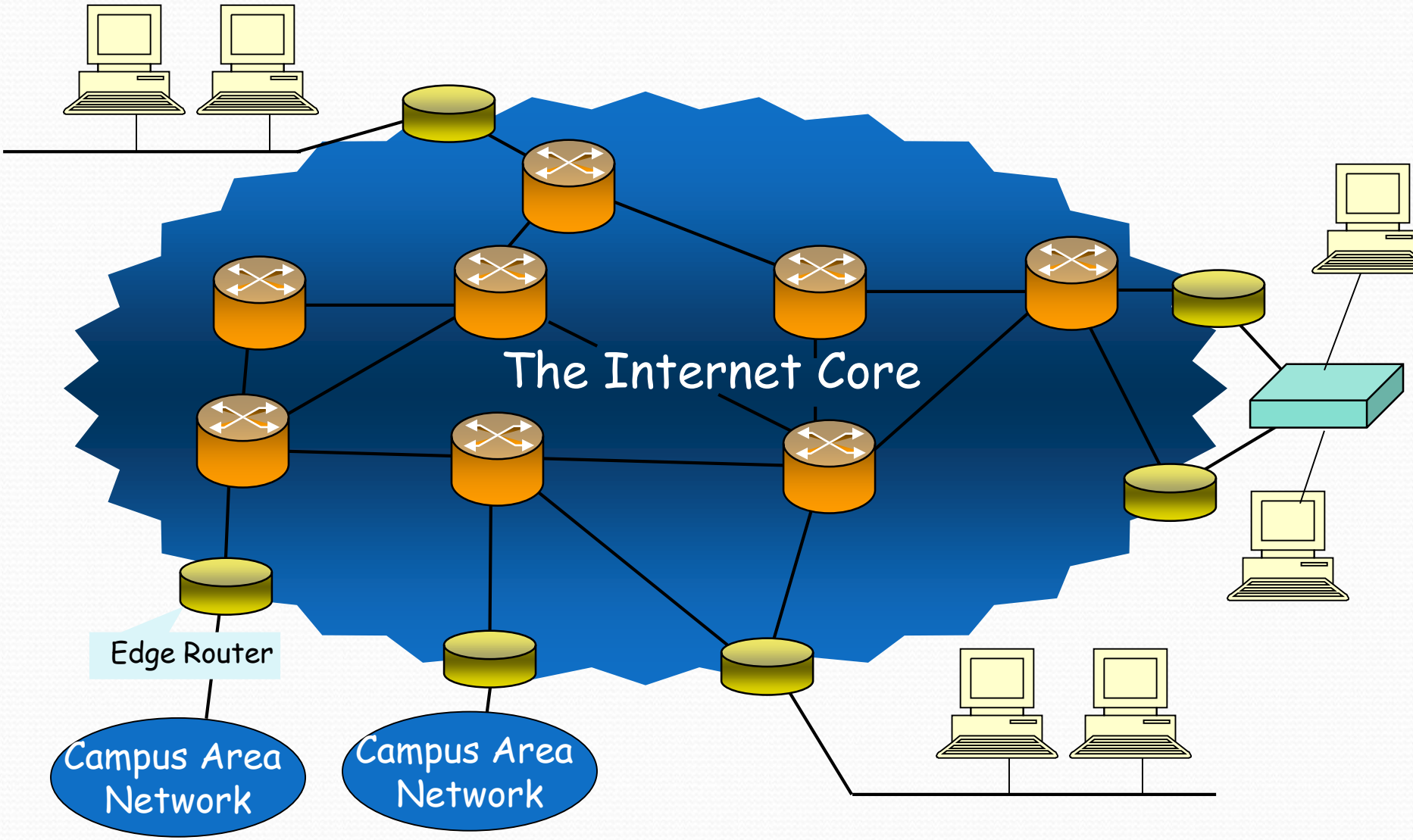
Outline

- Router overview
- IP lookup review (1-D packet classification)
- Binary prefix search
- Packet Classification (5 fields)
- Openflow (12 fields or more)
- partition-based Openflow packet classification

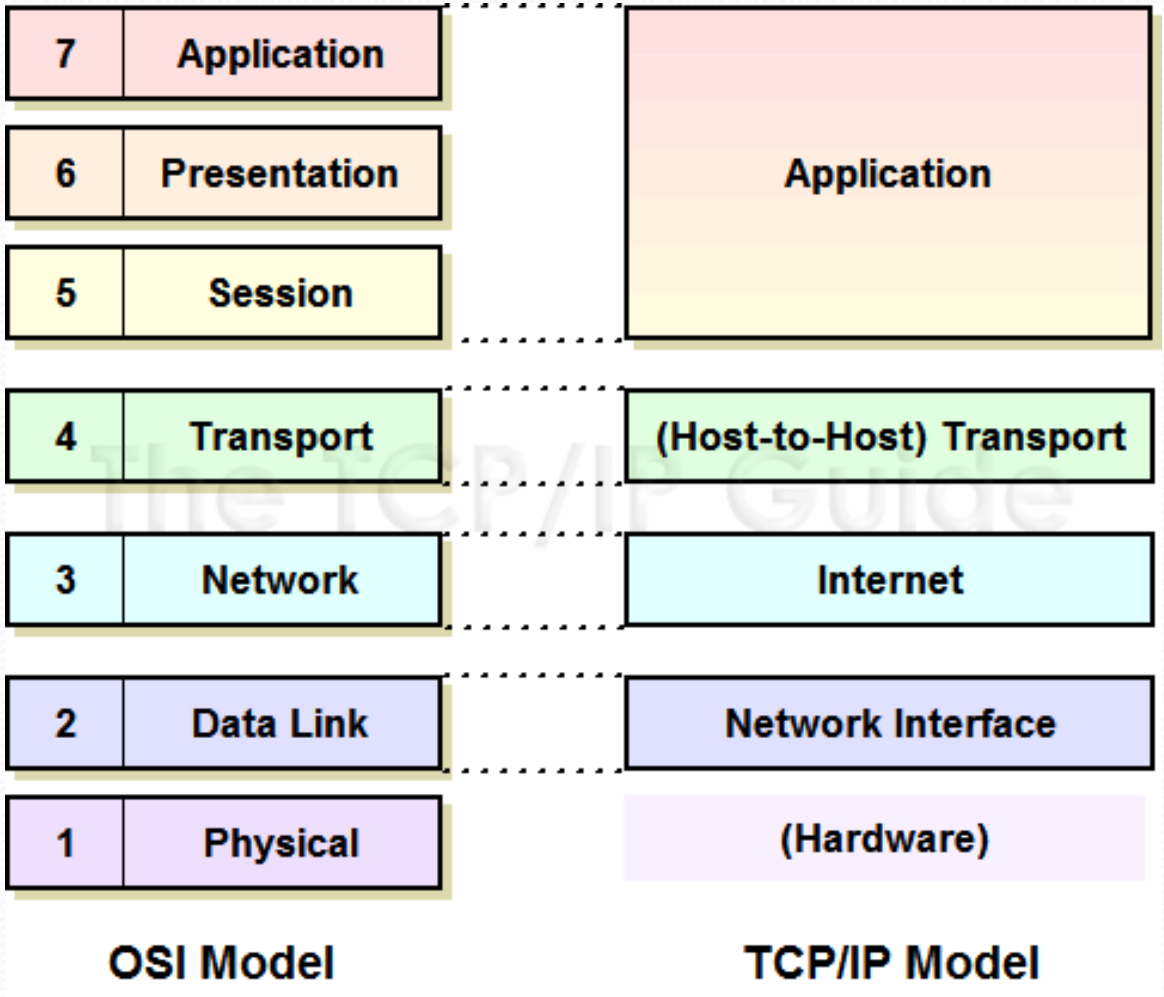
Router Overview

- Handle packet forwarding and routing protocol
- Traditionally, routers were implemented purely with software running on a PC based on a general purpose CPU with a number of interfaces.
- Such a device can receive packets on one of its interfaces, perform routing functions, and send packets out on another interface.
- As Internet traffics grow rapidly, type/size of routers changed since PC-based routers are limited by the performance of CPU and memory
- Fortunately, advances in silicon technology have made it possible to build hardware-based routers capable of handling high data rates.

Internet: Mesh of Routers



Network Protocols

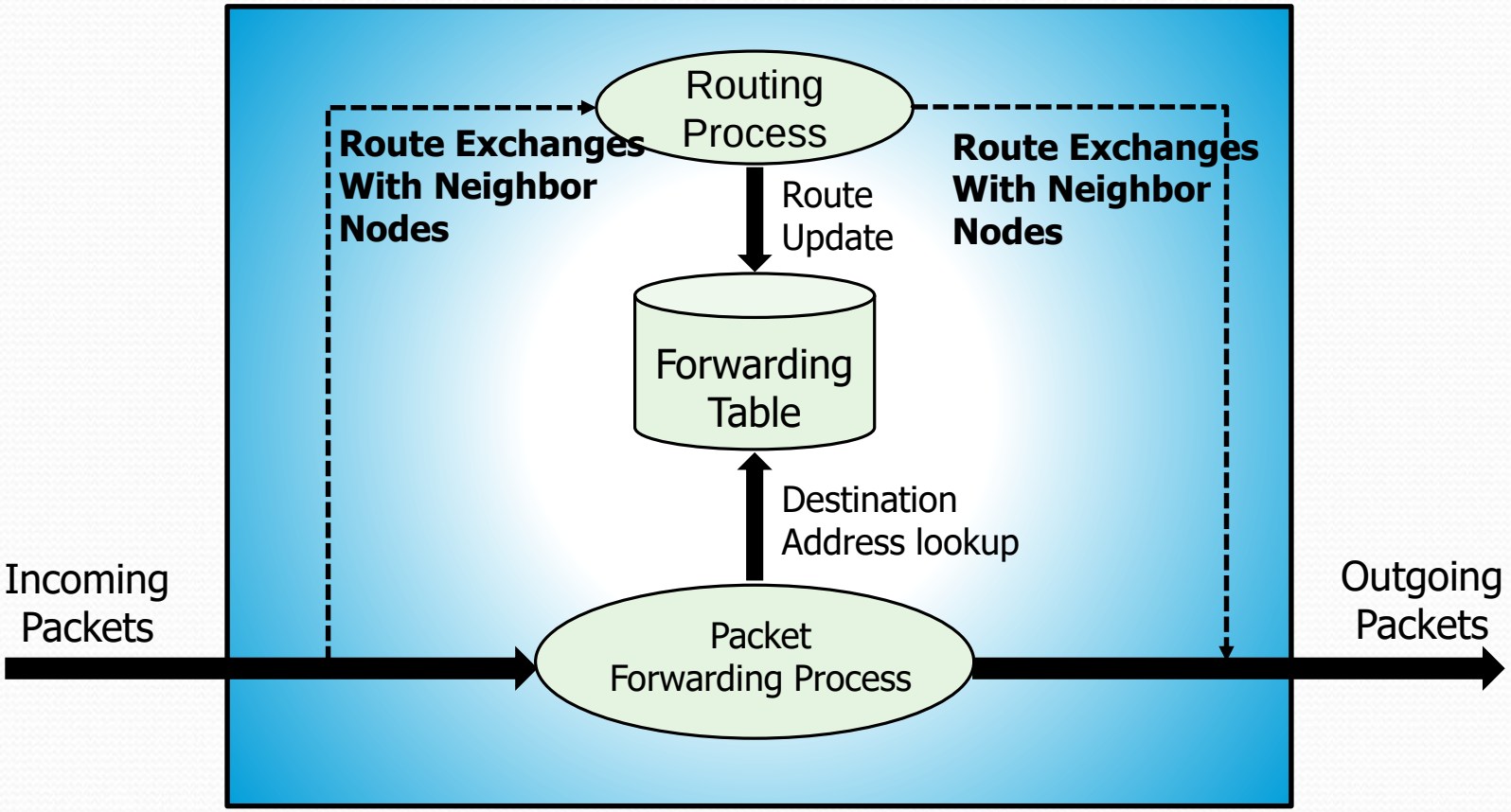


RFC 1812: Requirements for IPv4 Routers

- perform an IP datagram forwarding decision, called
  - **forwarding**,
  - ***routing lookup***,
  - ***IP lookup***,
  - ***longest prefix match***
- Must send the datagram out to the appropriate interface (called switching)
- Layer 2/3/4/5 switches

Functions of a Router

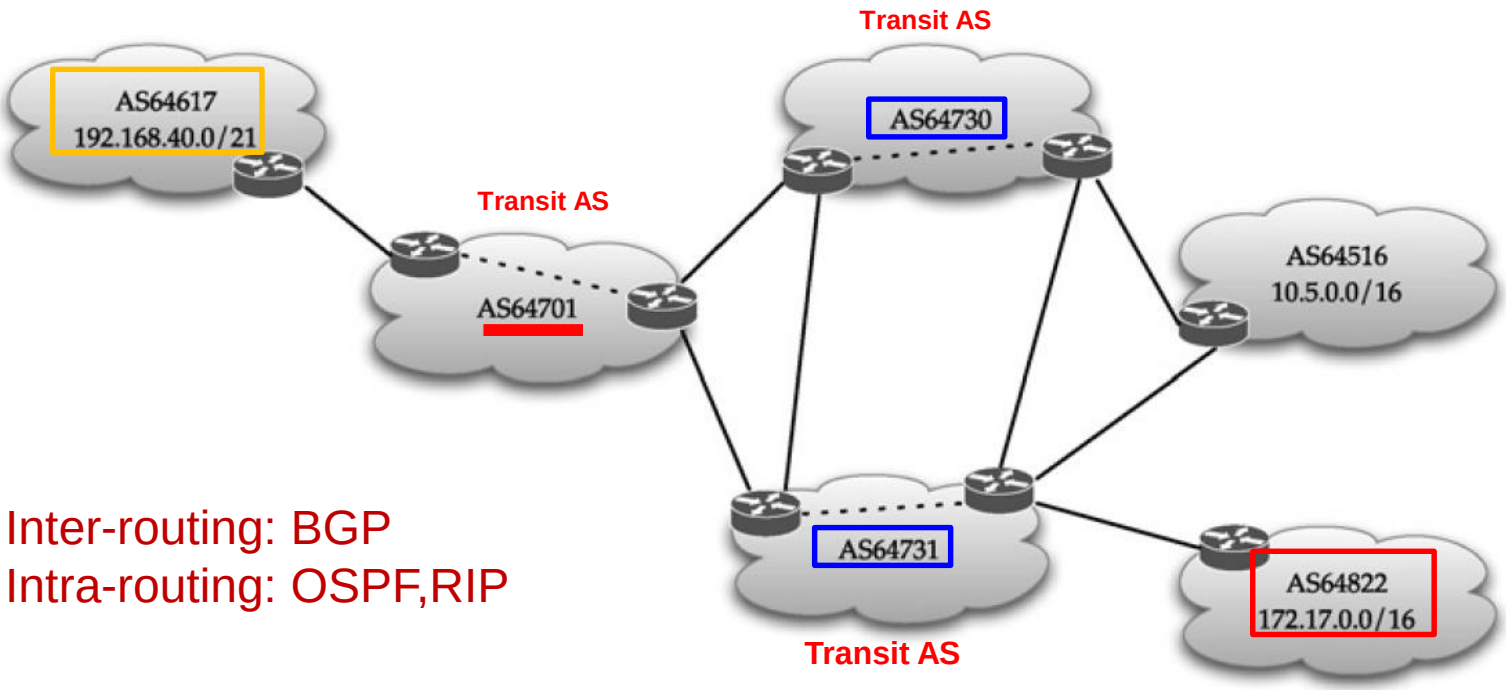
- Two fundamental tasks:
- Routing and Packet Forwarding



Functions of a Router

- Routing or Routing process
- Routing protocols are run to exchange information between neighboring routers
  - construct a view of the network topology which reflects network destinations that can be reached as identified through IP prefix-based network address blocks.
  - compute the best paths stored in a data structure called *a forwarding table*.

Routing process



Inter-routing: BGP
Intra-routing: OSPF,RIP

- at least one intermediate AS is default route-free
- If AS64701 maintains all default-free routers, the packets from AS64617 sending to a host of unallocated block can be dropped in AS64701.

Functions of a Router

- Packet forwarding
- Move a packet from an input interface ("ingress") of a router to the appropriate output interface ("egress") based on the information in the forwarding table.
- Since each packet arriving at the router needs to be forwarded, the performance of the forwarding process determines the overall performance of routers, the Internet.

Functions of a Router

- packet forwarding process is further divided into two subgroups: basic and complex
  - Basic forwarding defines the minimal set of functions a router should implement in order to transfer packets between interfaces.
  - Complex forwarding functions represent the additional processing required by the routers, depending on their deployment environments and their usage.

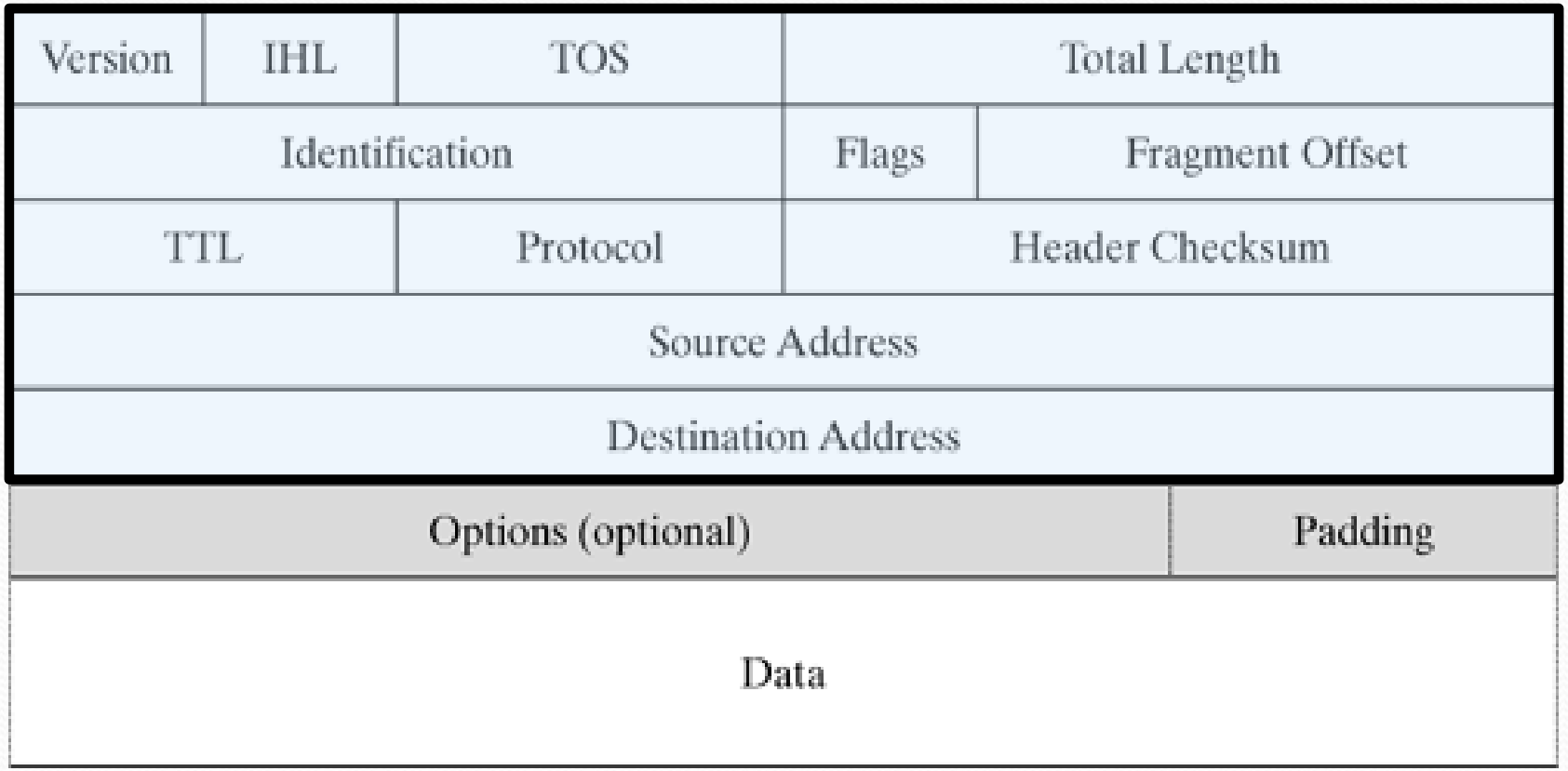
Basic Forwarding Functions

- IP Header Validation
- Packet Lifetime Control
- Checksum Recalculation
- Route Lookup
- Fragmentation
- Handling IP Options
- When there are routing or packet errors, routers use ICMP messages to communicate the information.

Basic Forwarding Functions

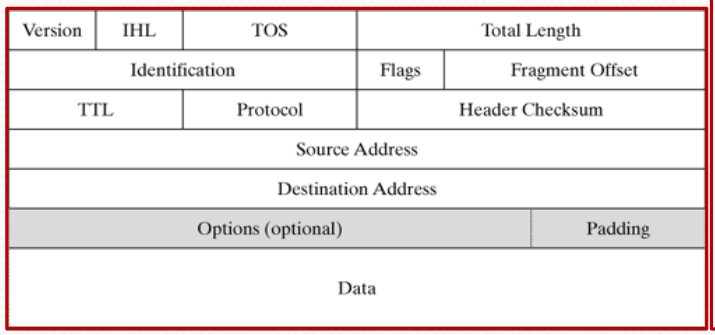
- *IP Header Validation*: Ensure only well-formed packets are processed further while the rest are discarded such as:
  - version number of the protocol is correct
  - header length of the packet is valid, and
  - the computed header checksum of the packet is same as the value of the checksum field in the packet header.

IP Header (at least 20 bytes)



IP CheckSum

- All 16-bit fields excluding checksum field are added together，the overflow bits are added back and then compute its complement
- 4500 0089 9713 0000 3e11 ~~chksm~~ 0a03 0001 e000 6464
- 4500+0089+9713+0000+3e11+0a03+0001+e000+6464
- = 2 6915
- 2+6915 = 6917
- 96e8 (complement)

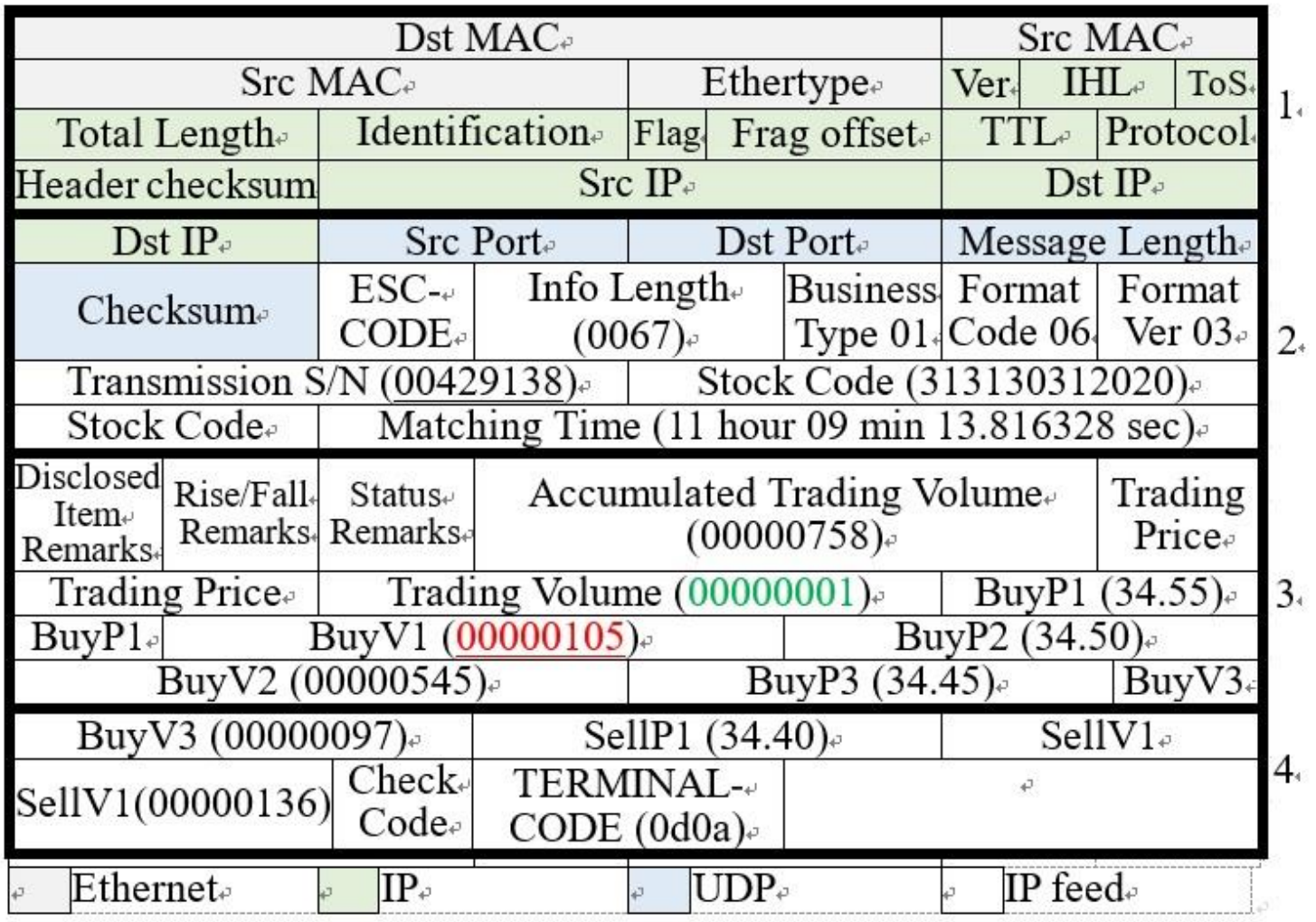


Version	IHL	TOS	Total Length
Identification		Flags	Fragment Offset
TTL	Protocol	Header Checksum	
Source Address			
Destination Address			
Options (optional)			Padding
Data			

```
Internet Protocol Version 4, Src: 10.3.0.1, Dst: 224.0.100.100
   0100 .... = Version: 4
   .... 0101 = Header Length: 20 bytes (5)
 > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   Total Length: 137  0089 = hex(137)
   Identification: 0x9713 (38675)
 > Flags: 0x0000
   Time to live: 62 = 3e = hex(62)
   Protocol: UDP (17) = 11 = hex(17)
   Header checksum: 0x96e8 [validation disabled]
   [Header checksum status: Unverified]
   Source: 10.3.0.1
   Destination: 224.0.100.100
```

WireShark packet extract tool

UDP


Dst MAC
Src MAC
Src MAC
Ethertype
Ver
IHL
ToS
1
Total Length
Identification
Flag
Frag offset
TTL
Protocol
Header checksum
Src IP
Dst IP
Dst IP
Src Port
Dst Port
Message Length
Checksum
ESC-CODE
Info Length (0067)
Business Type 01
Format Code 06
Format Ver 03
2
Transmission S/N (00429138)
Stock Code (313130312020)
Stock Code
Matching Time (11 hour 09 min 13.816328 sec)
Disclosed Item Remarks
Rise/Fall Remarks
Status Remarks
Accumulated Trading Volume (00000758)
Trading Price
Trading Price
Trading Volume (00000001)
BuyP1 (34.55)
3
BuyP1
BuyV1 (00000105)
BuyP2 (34.50)
BuyV2 (00000545)
BuyP3 (34.45)
BuyV3
BuyV3 (00000097)
SellP1 (34.40)
SellV1
SellV1(00000136)
Check Code
TERMINAL-CODE (0d0a)
4
Ethernet
IP
UDP
IP feed

TCP

Dst MAC										Src MAC						
Src MAC						Ethertype				ver	IHL			ToS		1
Total Length			Identification			Flag	Frag offset			TTL			Protocol			
Header checksum			Src IP							Dst IP						
Dst IP			Src Port			Dst Port				Sequence #						
Sequence #			Acknowledgement #							Len	Rsv	URG	ACK	PSH	RST	SYN	FIN	2
Window size			Checksun			Urgent pointer				8			=			
F	I		X		.	4		.		4			<SOH>			
9=195<SOH>35=D<SOH>34=000000<SOH>49=CLIENT1<SOH>																3
52=20180108-12:24:48.373<SOH>56=EXEC																4
UTOR<SOH>11=D0C4D14D1BC4<SOH>37=Z0bLJ<SOH>1=																5
8888885<SOH>55=071942<SOH>40=2<SOH>38=431<SOH>54																6
=1<SOH>44=000078200<SOH>59=0<SOH>60=20180108																7
-12:24:48<SOH>10000=1<SOH>10001=0<SOH>10002=																8
0<SOH>10004=N<SOH>10=178<SOH>																9

Ethernet IP TCP FIX new order

Basic Forwarding Functions

- *Packet Lifetime Control*: Routers must decrement the time-to-live (TTL) field in the IP packet header to prevent packets from **getting caught in the routing loops forever**.

- If the TTL value is zero or negative, the packet is discarded; an ICMP message is generated and sent to the original sender.

Basic Forwarding Functions

- *Checksum Recalculation*: Since the value of the TTL is modified, the header checksum needs to be updated.
- Instead of computing the entire header checksum again, it is more efficient to compute it incrementally; after all, the TTL value is always decremented by 1.

Basic Forwarding Functions

- *Route Lookup*: The destination address of the packet is used to search the **forwarding table** for determining the output port.
- The result of this search will indicate whether the packet is destined for the router
  - to an output port (unicast) or
  - to a set of multiple output ports (multicast).

Basic Forwarding Functions

- *Fragmentation*: It is possible that the maximum transmission unit (MTU) of the outgoing link is smaller than the size of the packet that needs to be transmitted.
- The packet would need to be split into multiple fragments before transmission.

Basic Forwarding Functions

- *Handling IP Options*: The presence of the IP options field indicates that there are special processing needs for the packet at the router.
- While such packets might arrive infrequently, a router nonetheless needs to support those processing needs.

Complex Forwarding Functions

- Security, different user requirements, and service guarantees based on different service level agreements (SLA)
  - Service differentiation example: watching a high-definition movie streaming directly over the Internet which requires (1) high bandwidth and (2) timely delivery of the data.
  - The router needs to distinguish such packets so that it can forward them earlier.
  - This results in the notion of differentiated services, and consequently requires that routers support a variety of mechanisms as follows:

Complex Forwarding Functions

- ***Packet Classification***
  - For distinguishing packets, a router might need to examine not only the destination IP address but also other fields such as source address, destination port, and source port, and protocol number.
  - Matching these headers against certain rules to find the matched rule whose actions are then applied.

Complex Forwarding Functions

- ***Packet Translation***
  - As the public IPv4 address space is being exhausted, there is a need to map several hosts to a single public address.
  - Thus, a router that acts as a gateway to a network needs to support network address translation (NAT).
  - NAT maps a public IP address into a set of private IP addresses and vice versa.
  - This requires a router to maintain a list of connected hosts and their local addresses and to translate the incoming and outgoing packets.

Complex Forwarding Functions

- ***Traffic Prioritization***
  - Guarantee a certain quality of service (QoS) to meet service level agreements, applying different priorities to different customers or data flows and providing a level of performance in accordance with the predetermined service agreements.
  - For example, the agreement might specify that a fixed number of packets must be delivered at a constant rate, necessary for real-time streaming multimedia applications such as IPTV, or real-time interactive applications such as VoIP

Control plane vs data plane

- Besides packet forwarding (i.e., data plane function), a router needs to ensure that the contents of the forwarding table reflect the *current network topology*.
- Routers also need to provide control plane and *management plane* functions. In particular, a router needs to handle:
  - *Routing Protocols*
  - *System Configuration*
  - *Router Management*

Control plane: *Routing Protocols*

- Routers need to implement different routing protocols, such as OSPF, BGP, and RIP for maintaining peer relationships by sending and receiving **route updates** from adjacent routers.
- These route updates are sent and received as normal IP packets.
- But the key **difference** between these packets and the packets that transit through the router is the **destination address=the router itself** for route update packets.
- Once the updates are received, the forwarding table is modified so that subsequent packets are forwarded to the correct outgoing links.

Control plane: *System Configuration*

- Network operators need to configure various administrative tasks:
  - Configuring interfaces,
  - Routing protocol keep alives,
  - Updating rules for classifying packets.
- Hence, a router needs to implement various functions for adding, modifying, and deleting these configuration data, as well as persistently storing them for retrieval later.

Control plane: *Router Management*

- Routers need to be monitored for continuous operations.
- These functions include supporting various management functions that are implemented using protocols such as simple network management protocol (SNMP).

Routing Table vs Forwarding Table

- ***routing table*** is constructed by routing algorithms of routing protocols, using information exchanged between routers.
  - Each entry in routing table maps **IP prefix** to **next hop**
- The ***forwarding table***, is consulted by the router to determine the output interface an incoming packet needs to be forwarded.
  - each entry in forwarding table maps ***IP prefix*** to **outgoing interface**
  - the entries might contain additional information such as the MAC address for the next hop and statistics about the number of packets forwarded through using the interface.

Routing Table vs Forwarding Table

- reasons to use two separate tables
  - forwarding table is optimized for searching an IP against many IP prefixes, routing table is optimized for calculating changes in the topology
  - as every packet needs to examine the forwarding table, it is implemented in a specialized hardware for high-speed routers. However, the routing tables are usually implemented in software.
- Ex:

(a) Routing table		(b) Forwarding table		
IP prefix	Next hop	IP prefix	Interface	MAC address
10.5.0.0/16	192.168.5.254	10.5.0.0/16	eth0	00:0F:1F:CC:F3:06

Performance of Routers

- Throughput: bits per second (bps)
  - how much data the router can transfer per second from input network interfaces to an output network interface.
- Throughput $T = P \times R$,
  - P = the number of ports or interfaces feeding the router and
  - R = the line rate of each port.
- For instance, a router containing 16 ports with each running at a line rate of 40 Gbps has a throughput of 640 Gbps.

Performance of Routers

- As routers forward packets, it is more important to know how many packets they are capable of forwarding in a second, which is referred to as ***packets per second*** (***pps***).
- For instance, a router throughput of 640 Gbps could mean **packets of size** 40 bytes forwarded at 2 billion pps or packets of size 80 bytes forwarded at 1 billion pps.

Performance of Routers

- What should be the packet size used?
- In a decade-old study, the average packet size was found to be 300 bytes
- In recent observations, commonly seen sizes are
  - 40 bytes = 20 bytes (IP header) + 20 bytes (TCP header), ex. TCP acknowledgments,
  - 576 bytes (RFC 879, which is now outdated),
  - 1500 bytes (Ethernet MTU size),
  - 1300 bytes (VPN software), 64 bytes.
- If a router is designed with any of these sizes other than the smallest size, it might not be able to sustain a long sequence of shorter packets.
- Thus, most use the minimum of 40 bytes as the standard packet size for such assessment.

Types of Routers

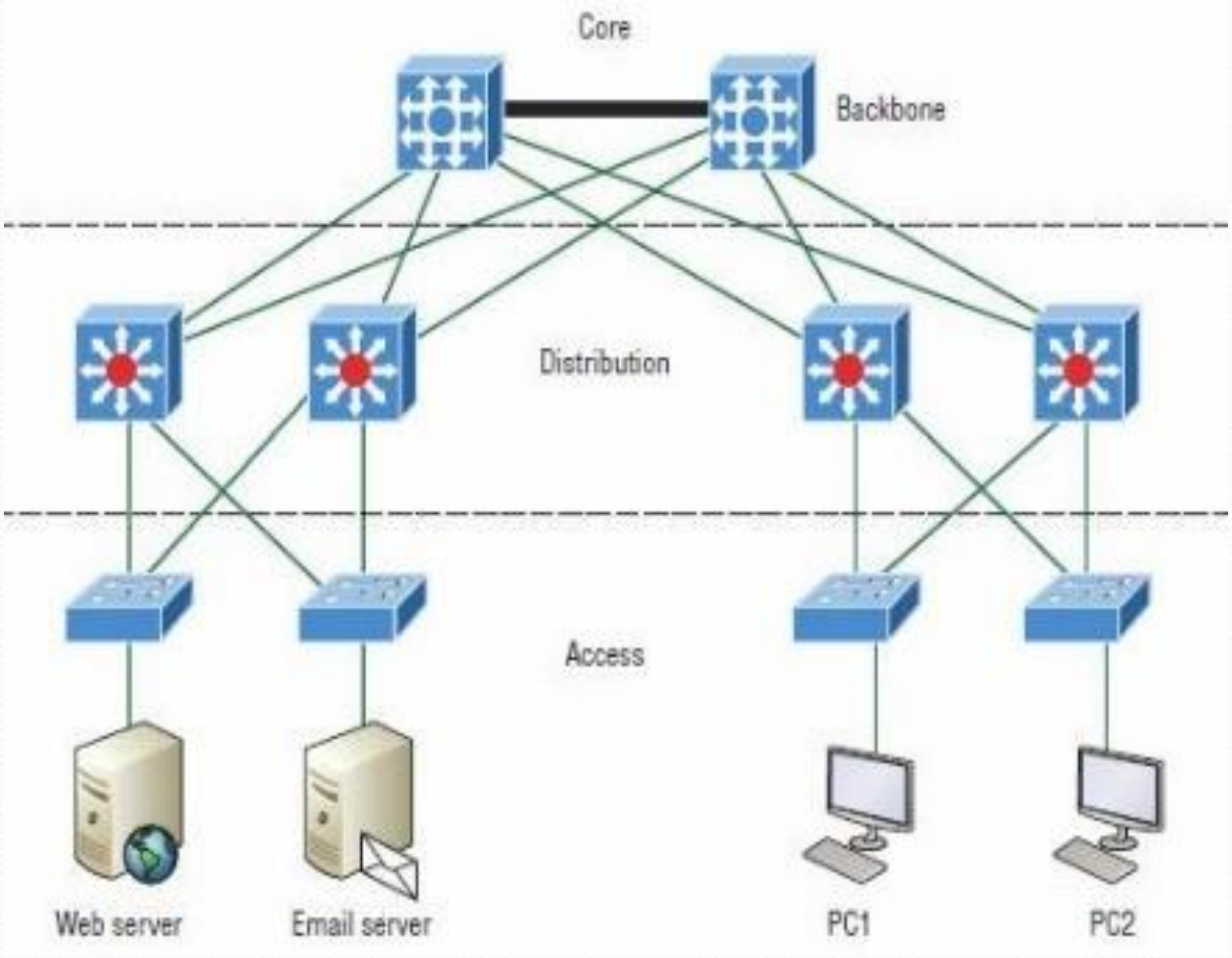
- Routers can be of different complexity based on
  - where in the network they are deployed
  - how much traffic they need to sustain.
- Naturally, this means that routers can be of different types.
- three types of routers: core routers, edge routers, and enterprise routers
- their requirements will be outlined

三層網路架構圖

Core
(核心層)

Aggregat
e
Distribute
(匯集層)

Access
(存取層)

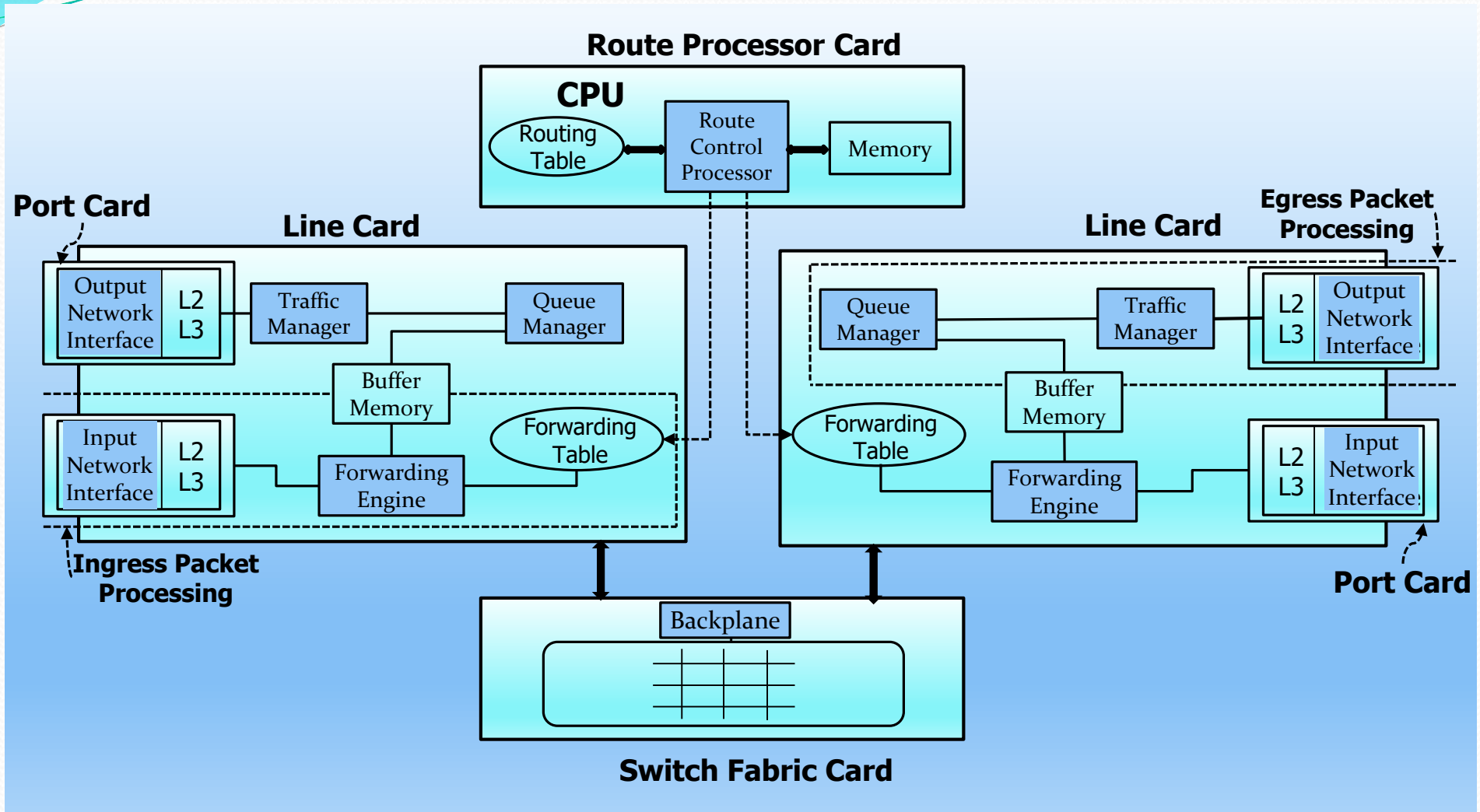


Elements of a Router

- router can be viewed from two different perspectives:
  - *Functional* perspective: logically viewed as a collection of modules where each module implements a set of related functions to achieve the overall goal of forwarding packets
  - *Architectural* perspective: considered as an interconnection of different types of cards running specialized software and How the functional modules are implemented in practice.

Elements of a Router

- *From functional* point of view: A router can be divided into several modules. These modules implement the various requirements of a router.
- A generic router consists of six major functional modules: (1) network interfaces, (2) forwarding engine,(3) queue manager, (4) traffic manager, (5) backplane, and (6) route control processor.
- These functional modules are shown in the following figure.


Route Processor Card
CPU
Routing Table
Route Control Processor
Memory
Port Card
Line Card
Output Network Interface
L2 L3
Traffic Manager
Queue Manager
Buffer Memory
Input Network Interface
L2 L3
Forwarding Table
Forwarding Engine
Ingress Packet Processing
Line Card
Egress Packet Processing
Queue Manager
Traffic Manager
L2 L3
Output Network Interface
Buffer Memory
Forwarding Table
Forwarding Engine
L2 L3
Input Network Interface
Port Card
Backplane
Switch Fabric Card

Network Interface

- contain ports connecting to physical network links
- A port terminates a physical link and serves as entry and exit points for incoming/outgoing packets.
- specific to a particular type of network physical medium. Ex. an Ethernet or a SONET. (400GE,100GE,100GE DWDM,40GE,10GE,10GE OTN,10GE DWDM,1000M), 10G SFP+
- network interface provides several functions.
  - understand various data link protocols and decapsulate the incoming packets by stripping the Layer 2 (L2) headers.
  - extract the IP headers, i.e., the Layer 3 (L3) headers, and sends them to the forwarding engine for route lookup while the entire packet is stored in memory.
  - Collectively, this processing is referred to as L2/L3 processing.
  - Further, it provides the functionality of encapsulating L2 headers before the packet is send out on the link.

Forwarding Engine

- Decide to which network interface incoming packet should be forwarded by a route lookup function.
- When a port receives a packet, it de-encapsulates L2 headers and sends entire IP packet, or just the packet header, to the forwarding engine.
- Route lookup can be implemented in custom hardware or software running on a commodity cpu.
- Depending on the architecture, the lookups can occur in the custom hardware or in a local route cache in the line card.
- To provide QoS guarantees, forwarding engines may need to classify packets into predefined service classes.

Queue Manager

- Provide buffers for temporary storage of packets when an outgoing link from a router is overbooked.
- When these buffer queues overflow due to congestion in the network, the queue manager selectively drops packets.
- Need to manage the occupancy of the queue and implement policies about which packets to drop when the queues are about to be fully occupied.

Traffic Manager

- prioritize and regulate the outgoing traffic, depending on the desired level of service.
- Necessary as routers carry traffic from different subscribers to ensure they get the level of service for which they pay.
- Shape the outgoing traffic to the subscriber according to the service level agreement.
- When receiving traffic from a subscriber, the traffic manager ensures that it does not accept more than what is specified in the contract.
- Sometimes the functionality of the queue manager and the traffic manager are merged into a single component.

Backplane

- Provide connectivity for the network interface card so that packets from an incoming network interface can be transferred to the outgoing network interface card.
- The backplane can be either ***shared***, where only two interfaces can communicate at any instant, or ***switched***, where multiple interfaces can communicate simultaneously.
- The aggregate bandwidth of all the attached network interfaces defines the bandwidth required for the backplane.

Route Control Processor

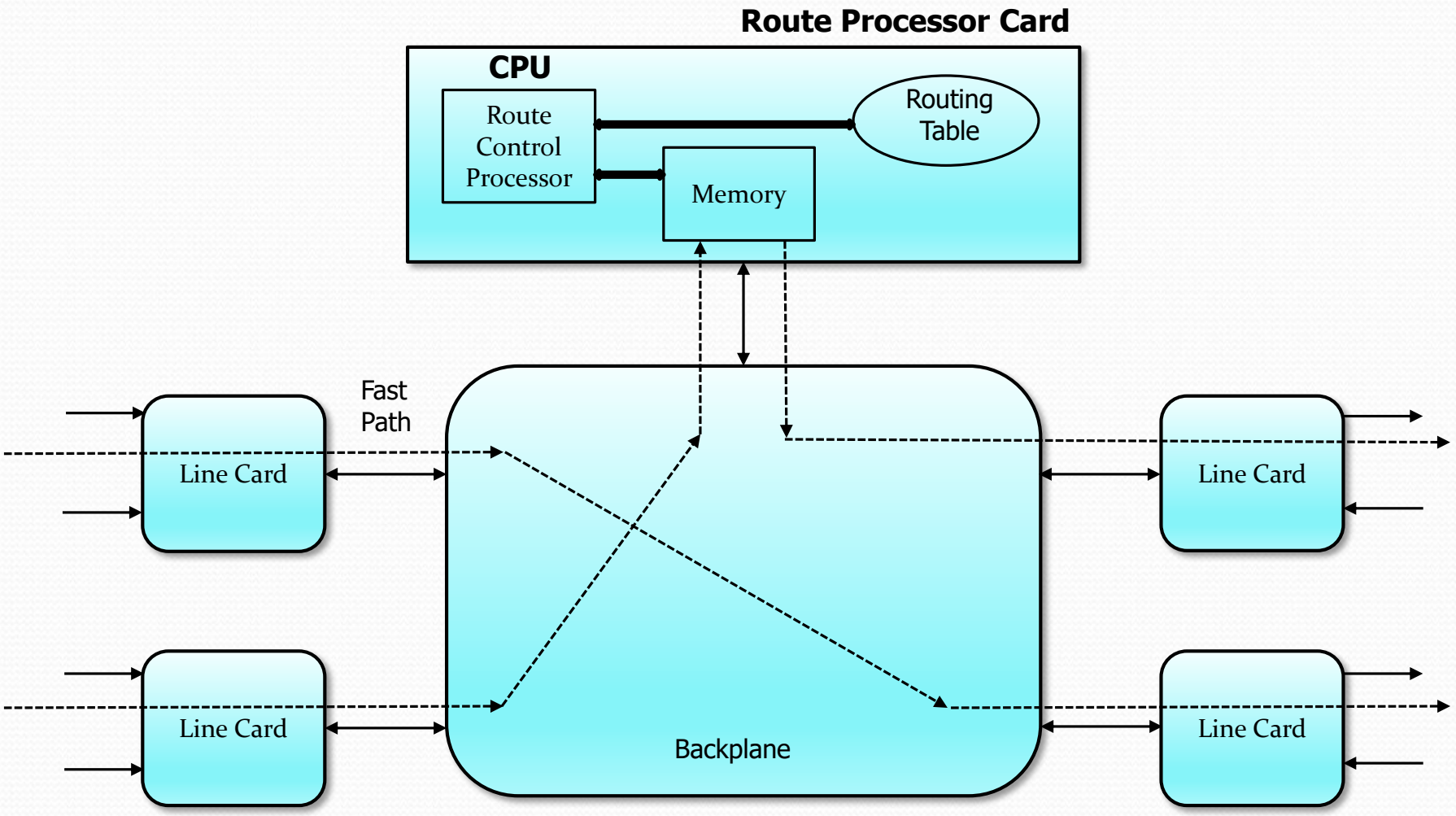
- Implementing and executing routing protocols for maintaining a routing table that is updated whenever a route change occurs.
- Based on the contents of the routing table, the forwarding table is computed and updated.
- Run the software to configure and manage the router.
- Performs complex packet-by-packet operations like errors during packet processing.
  - For example, it handles any packet whose destination address cannot be found in the forwarding table in the line card by sending an ICMP packet to its source of origin indicating the error.
  - These functionalities are typically implemented in software running on a general-purpose microprocessor.

Slow path vs Fast path

- Tasks performed are categorized into *time-critical* and *non-time-critical* operations depending on their frequency, called *fast path* and *slow path*.
- Time-critical operations affect the majority of the packets and need to be highly optimized in order to achieve gigabit forwarding rates (Hardware).
  - grouped into *header processing and forwarding*.
    - Header processing include packet validation, packet lifetime control, and checksum calculation,
    - Forwarding include IP lookup, packet classification for service differentiation, packet buffering, and scheduling.

Slow path vs Fast path

- Non-time-critical tasks are typically performed on packets for maintenance, management, and error handling.
  - Processing of data packets that lead to errors in fast path and generation of ICMP packets to inform the originating source of the packets
  - Processing of routing protocol keep-alive messages from adjacent neighbors and sending of these messages to the neighboring routers
  - Processing of incoming packets that carry route table updates and sending messages to neighboring routers when network topology changes
  - Processing of packets pertaining to management protocols, such as SNMP, and the associated replies


Route Processor Card
CPU
Route Control Processor
Routing Table
Memory
Fast Path
Line Card
Line Card
Line Card
Line Card
Backplane

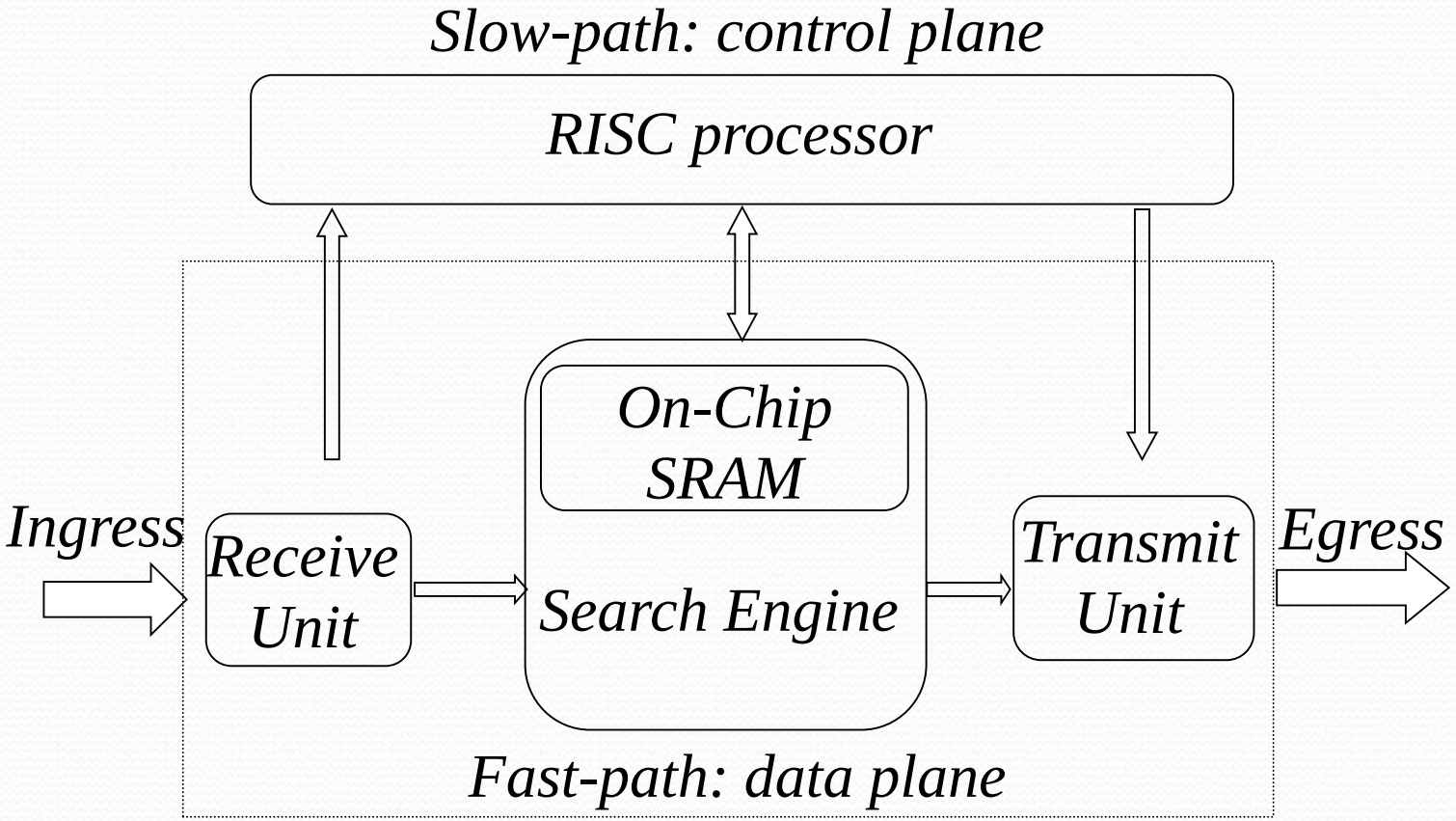
PACKET FORWARDING

- Determine next-hop IP address for the incoming packet and decide which output port and network interface should be used to send the packet. The result of the lookup could lead to three possibilities
  - *Local*: If packet is destined for the router's local IP address, it is delivered to the route control processor. i.e., routing protocol keep-alives and route-updates.
  - *Unicast*: Packet is delivered to a single output port on a network interface, either a next-hop router or to the ultimate destination.
  - *Multicast*: packet is delivered to a set of output ports on the same or different network interfaces, based on multicast group membership, which is maintained by the router.

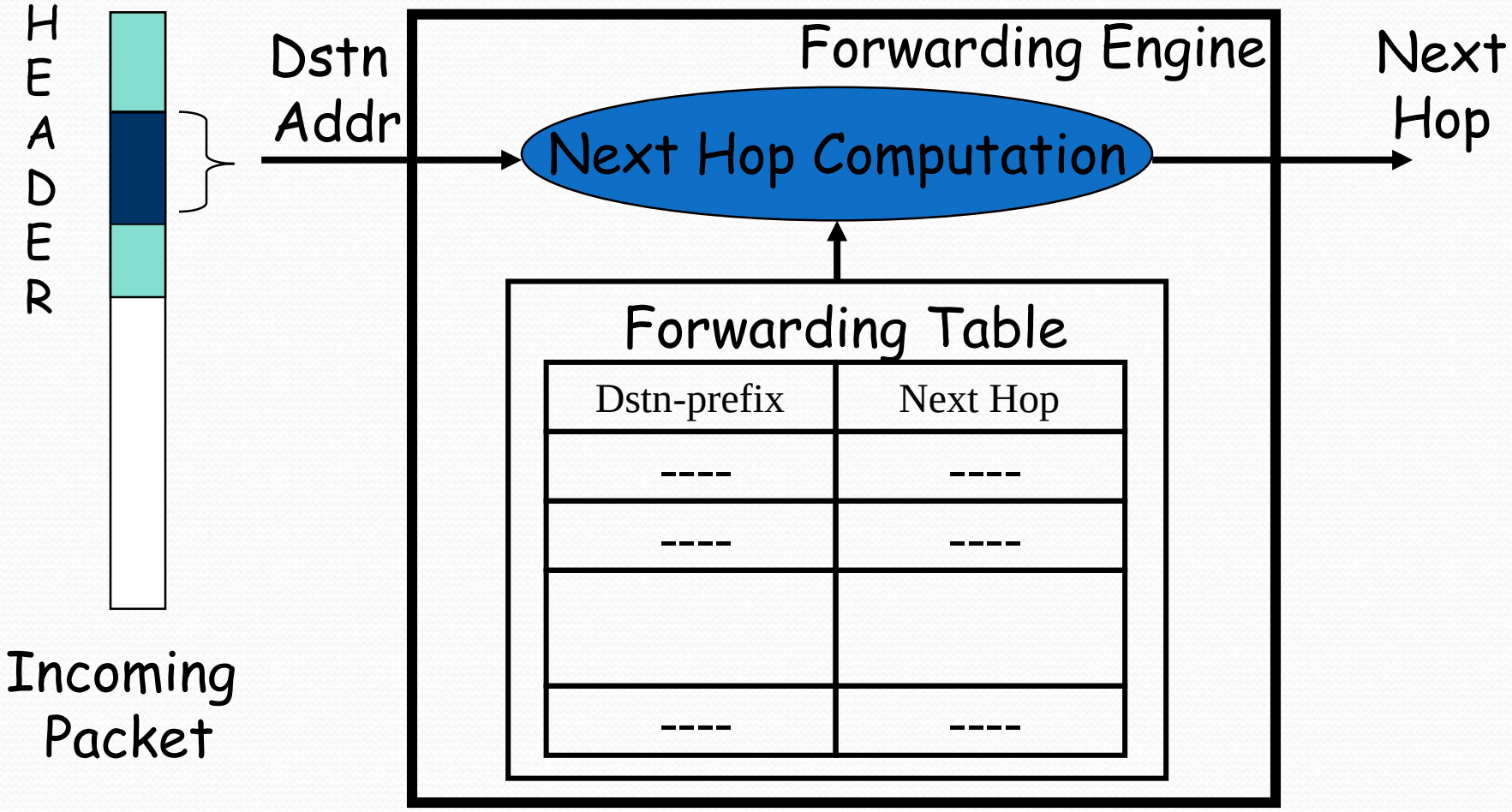
PACKET CLASSIFICATION

- isolate different classes/types of IP traffic, based on information carried in the packet.
- Depending on packet type, an appropriate action is applied against a set of rules (*classifier*).
- *5-tuple:* source/destination address, source/destination port, protocol
- The source and destination addresses identify the participating endpoints, the protocol flags identify the type of payload, and the source and destination ports identify the application (assuming the payload is TCP or UDP).
- should be fast enough to keep up with the line rate by using fast and efficient algorithms.

Router Design Model



Lookup in an IP Router



Unicast destination address based lookup

BGP Table Data

- **BGP Table Data**

  last updated at Wed, **23 Nov 2011** 15:12:48 GMT
- **IPv4 BGP Reports**
  - AS131072 APNIC R&D 385,044
  - AS6447 Route-Views.Oregon-ix.net 396,386
- **IPv6 BGP Reports**
  - AS131072 APNIC R&D 7,616
  - AS6447 Route-Views.Oregon-ix.net 7,581

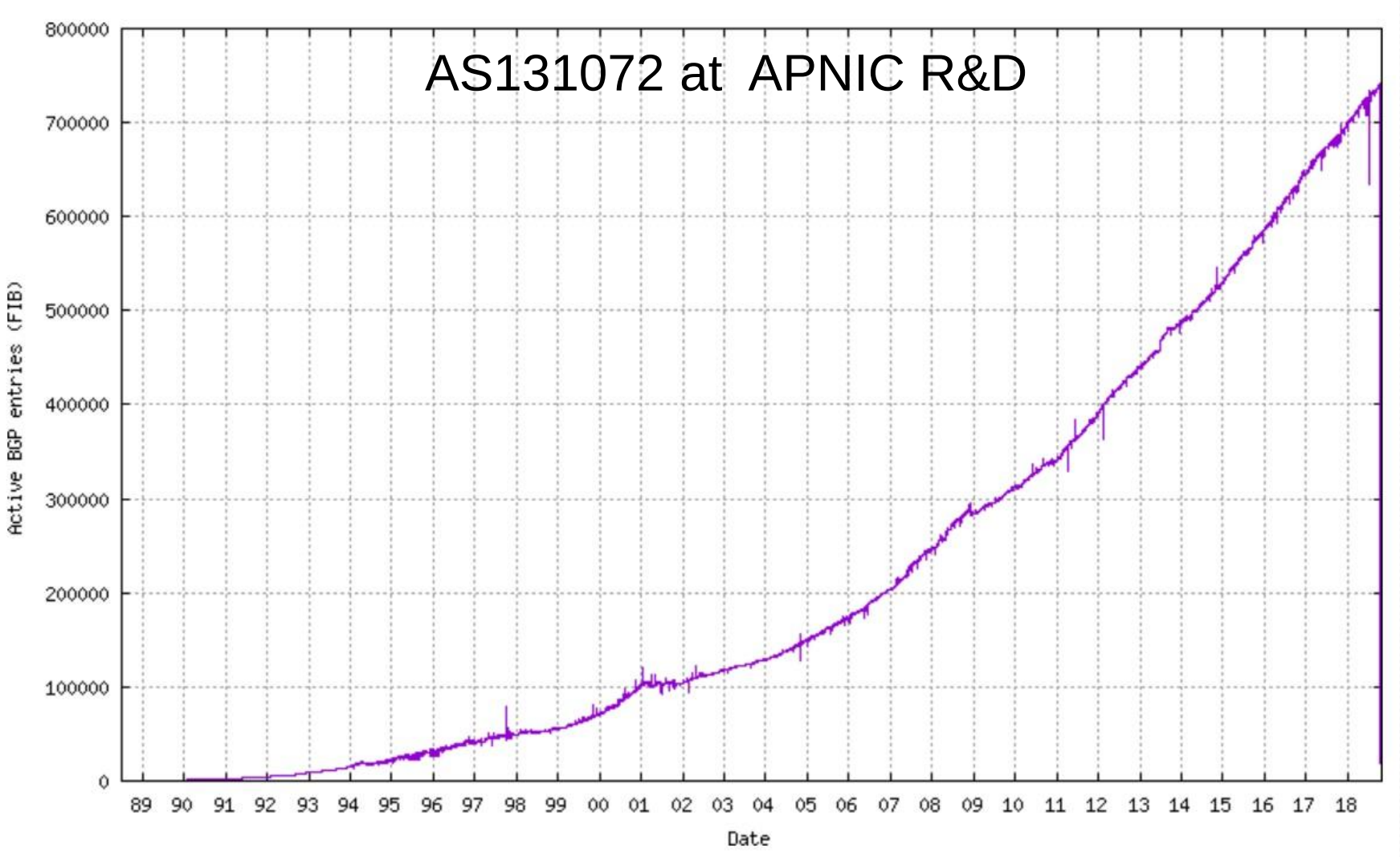
BGP Table Data

- last updated at Sun, **28 Oct 2018** 12:20:11 GMT
- **IPv4 BGP Reports**
  - AS131072 APNIC R&D 740,366
  - AS6447 Route-Views.Oregon-ix.net 777,257
- **IPv6 BGP Reports**
  - AS131072 APNIC R&D 59,965
  - AS6447 Route-Views.Oregon-ix.net 63,186

Routing table example

- 1.5.0.0/16
- 1.9.0.0/16
- 1.9.2.0/24
- 1.9.4.0/22
- 1.9.12.0/24
- 1.11.0.0/21
- 1.11.8.0/21
- 1.11.16.0/21
- 1.11.24.0/21
- 1.11.32.0/21
- 1.11.40.0/21
- 1.11.48.0/21
- 1.11.56.0/21
- 1.11.64.0/21
- 1.11.72.0/21
- 1.11.80.0/21
- 1.11.88.0/21
- 1.11.128.0/17
- 1.12.0.0/14
- 1.12.0.0/24
- 1.12.1.0/24
- 1.21.0.0/16
- 1.22.0.0/23
- 1.22.4.0/23
- 1.22.6.0/23
- 1.22.8.0/23
- 1.22.12.0/23
- 1.22.14.0/23
- 1.22.16.0/23
- 1.22.18.0/23

Active BGP entries (FIB)



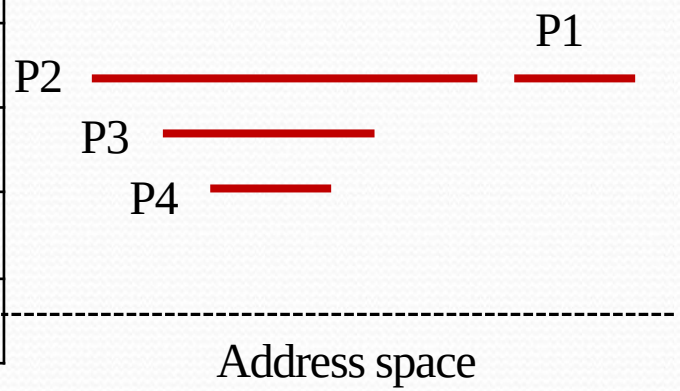
Router

- A memory hungry search application
- For a link of 10Gbps and packets of minimum size 40 bytes, $10 \times 10^9 / 8 \times 40 = \mathbf{31.25 \times 10^6}$ packets per second
- Router speed depends on the number of mem accesses for each lookup operation, i.e., on the speed of memory
- IPv6 is four times wider than IPv4 addresses
- Negative Impact: 4 × # of mem accesses for 32-bit CPU
- IPv6 routers is four times slower than IPv4 routers?
  - Not really, but possible
- Pipeline design may be a good solution

Example Forwarding Table

- Longest prefix match (LPM), not exact match
- Properties: prefixes are either **disjoint** or **enclosing** (one completely covers another)
- Prefix enclosure makes (1) sorting prefixes and (2) binary searching prefixes ***difficult***.
- So, trie based schemes emerge naturally

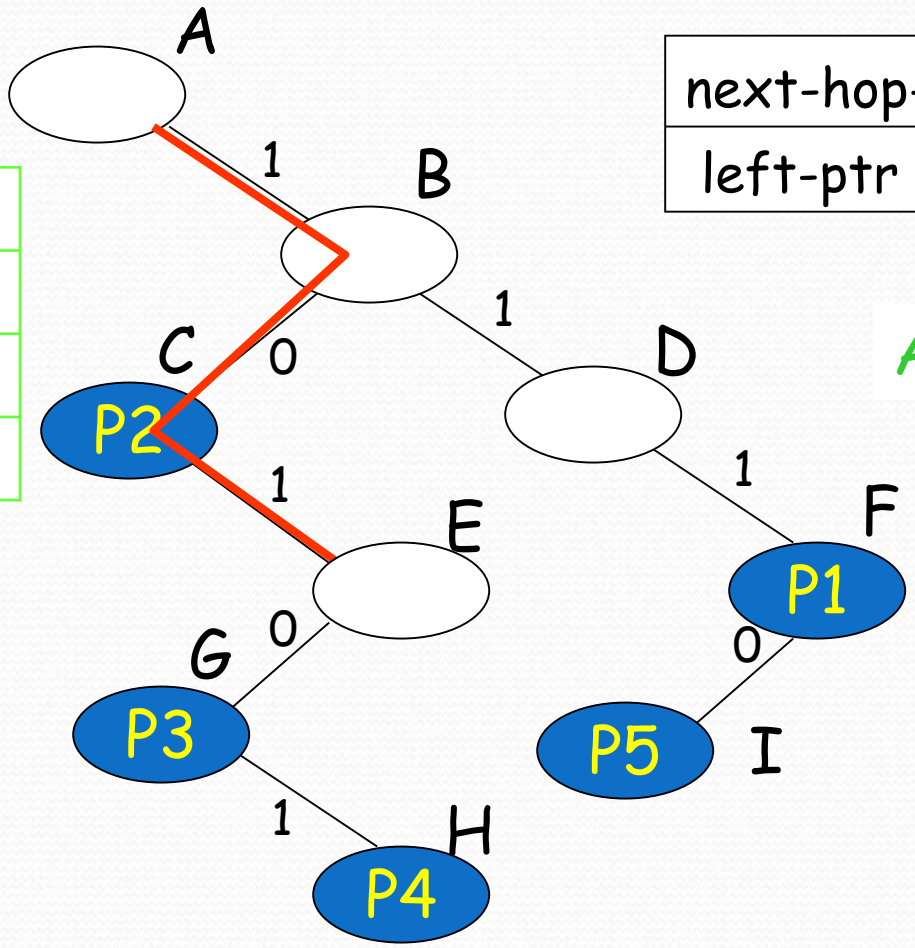
	Prefix	Next-hop
P1	111*	H1
P2	10*	H2
P3	1010*	H3
P4	10101	H4



Binary Trie (Radix Trie)

Lookup 10111

P1	111*	H1
P2	10*	H2
P3	1010*	H3
P4	10101	H4



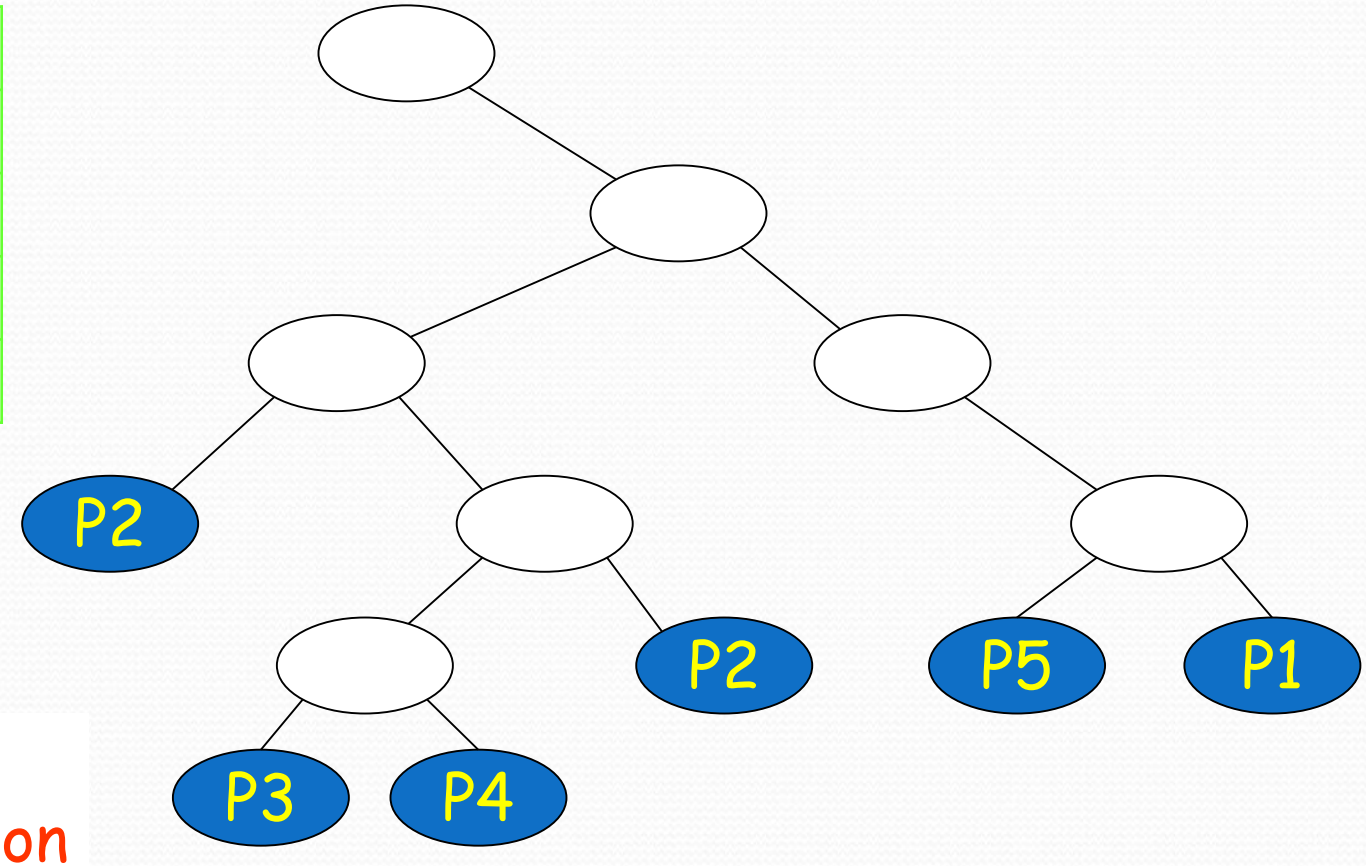
Trie node

next-hop-ptr (if prefix)	
left-ptr	right-ptr

Add P5=1110*

Binary Trie: Leaf Pushing

P1	111*	H1
P2	10*	H2
P3	1010*	H3
P4	10101	H4
P5	1110	H5

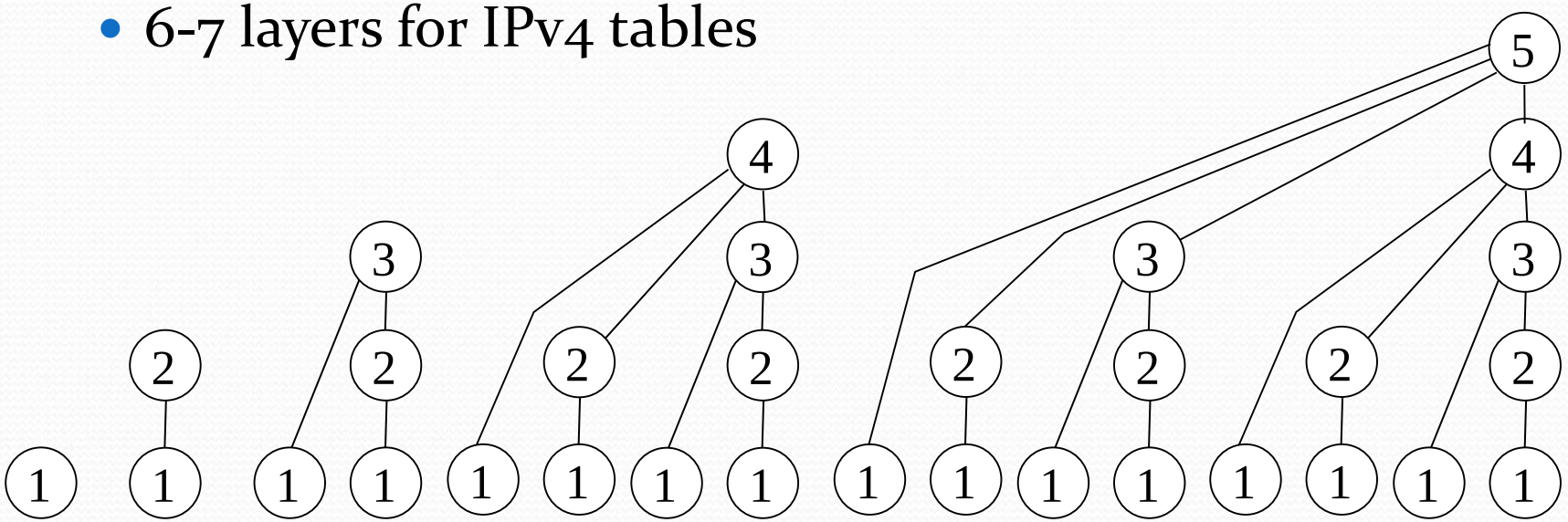


Disjoint,
but duplication

Basic Data Structures for IP lookups

Prefix properties

- *The most specific prefixes (MSP)*:
  - The prefixes that do not cover any others.
  - Disjoint, so can be put in an array for binary search
- Grouping prefixes in layers based on MSP.
  - 6-7 layers for IPv4 tables

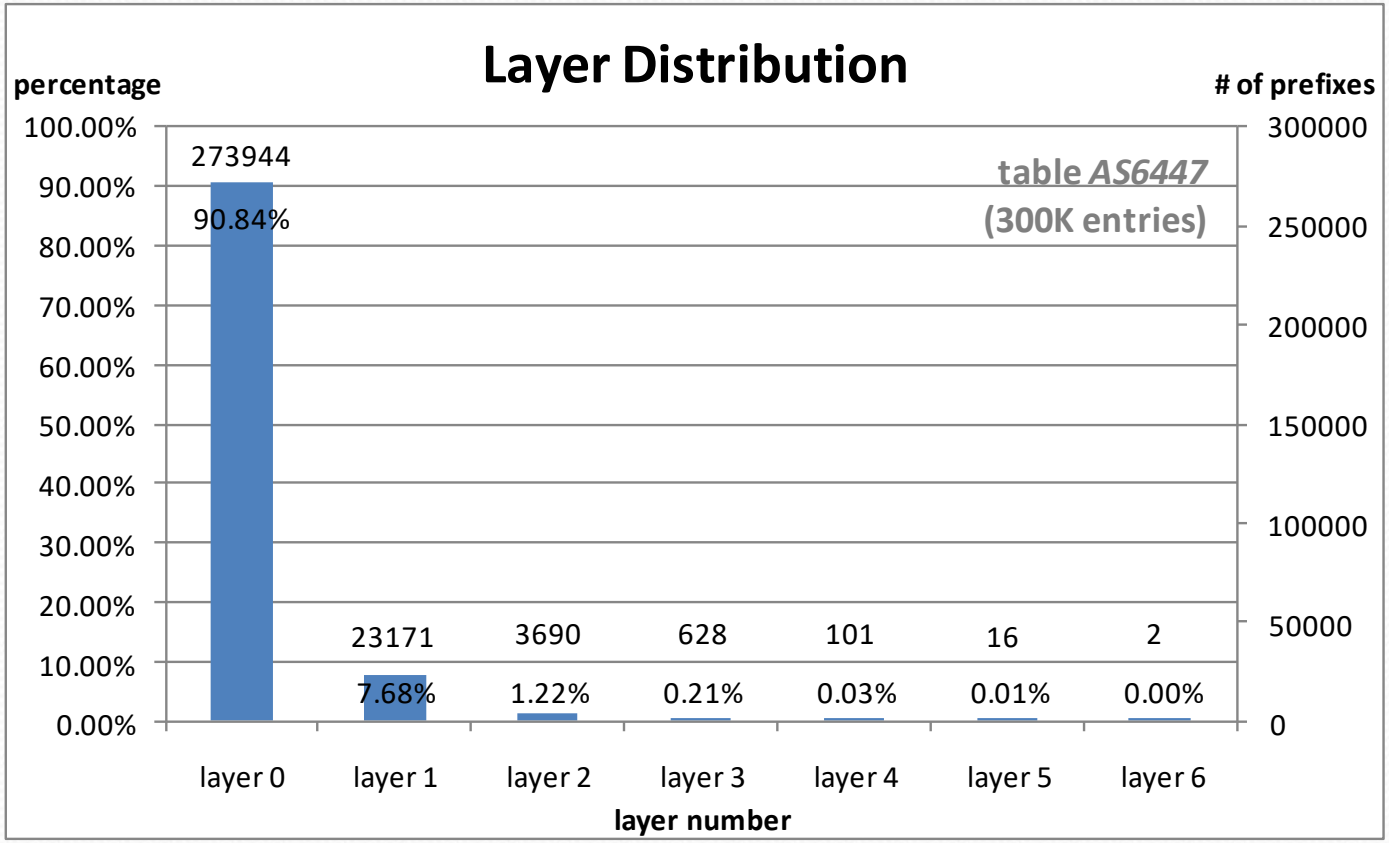


Prefix Enclosure property

Database (year-month)	AS6447 (2000-4)	AS6447 (2002-4)	AS6447 (2005-4)
number of prefixes	79,530	124,798	163,535
Level-0	73,891(92.9%)	114,745 (91.9%)	150,245 (91.9%)
Level-1	4,874 (6.1%)	8,496 (6.8%)	11,135 (6.8%)
Level-2	642 (0.8%)	1,290 (1%)	1,775 (1.1%)
Level-3	104 (0.1%)	235 (0.2%)	329 (0.2%)
Level-4	17	29	45
Level-5	2	3	6

Prefix Enclosure property

- Layer distribution



Prefix Enclosure Analysis (1/2)

Routing Table		AS6447 (2005-4)	AS6447 (2009-7)	AS6447 (2011-5)
size		163,535	301,552	369,394
Layer	0	150,245 (91.9%)	273,944 (90.8%)	334,445 (90.5%)
	1	11,135 (6.8%)	23,171 (7.7%)	28,930 (7.8%)
	2	1,775 (1.1%)	3,690 (1.2%)	4,870 (1.3%)
	3	329 (0.2%)	628 (0.2%)	926 (0.3%)
	4	45	101	177
	5	6	16	30
	6	0	2	12
	7	0	0	4

Prefix Enclosure Analysis (2/2)

- After split table and small push

Routing Table		AS6447 (2005-4)		AS6447 (2009-7)		AS6447 (2011-5)	
size		163,535		301,552		369,394	
length		9-24	25-32	9-24	25-32	9-24	25-32
Layer	0	7,071	3,488	271,548	5,736	333,034	5579
	1	496	67	17,081	28	20,884	21
	2	54	0	1,488	0	1,859	0
	3	6		88		111	
	4	0		2		5	

Prefix

Forwarding table example

	Prefix	Next-hop
P1	111*	H1
P2	10*	H2
P3	1010*	H3
P4	10101	H4

- P1 is disjoint from the other three prefixes.
- $P2 \supseteq P3 \supseteq P4$
- Longest prefix match(LPM), not exact match
- enclosure makes (1) sorting prefixes and (2) binary searching prefixes difficult

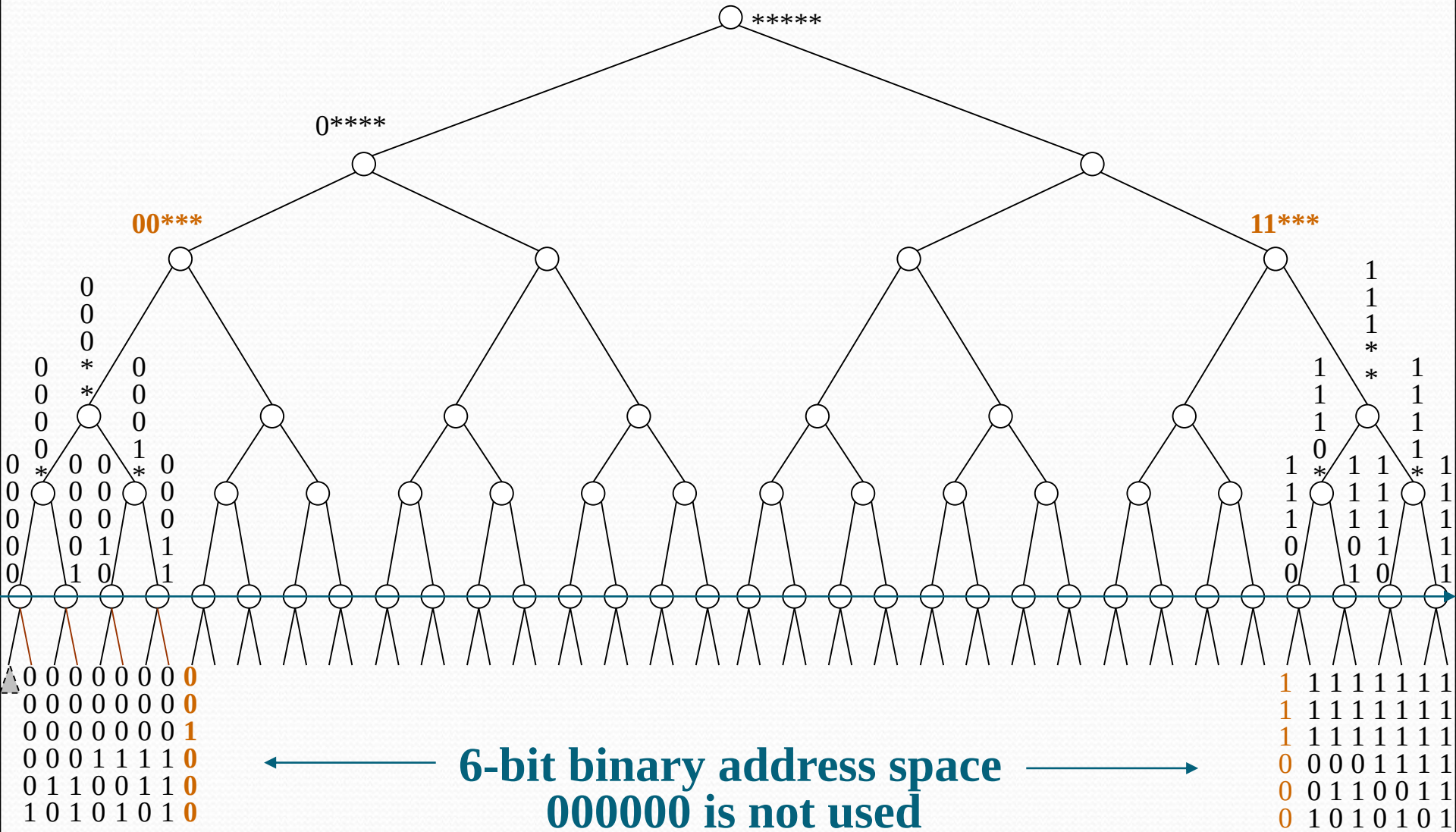
Prefix format

- ***Length* (32+6 bits)**: **$b_{n-1}\ldots b_0/\ell$ (ℓ is prefix length)**
  - In IPv4, ***d3.d2.d1.d0/ℓ*** , ***140.116.82.36/24*** .
- ***Mask* (32+32 bits)**: **$b_{n-1}\ldots b_0/m_{n-1}\ldots m_0$ (prefix length is ℓ)**
  - $m_j = 1$ for all $n-1 \geq j \geq n-\ell$, and $m_j = 0$ otherwise.
  - $d_3.d_2.d_1.d_0/\ m_3.m_2.m_1.m_0$, ***140.116.82.36/1...100000000***
- ***Ternary* (32 ternary bits)**: **$b_{n-1}\ldots b_{n-\ell+1}$*…* (length is ℓ)**
  - For TCAM memory
  - $b_j = 0$ or 1 for $n-1 \geq j \geq n-\ell$.
  - If t_k is *, then t_j must also be * for all $j < k$.
  - A single don't care bit can be used to denote a series of don't care bits, e.g., 1* denotes 1**** in the 5-bit address space. ***140.0.0.0/8 = 10001100****

Prefix format

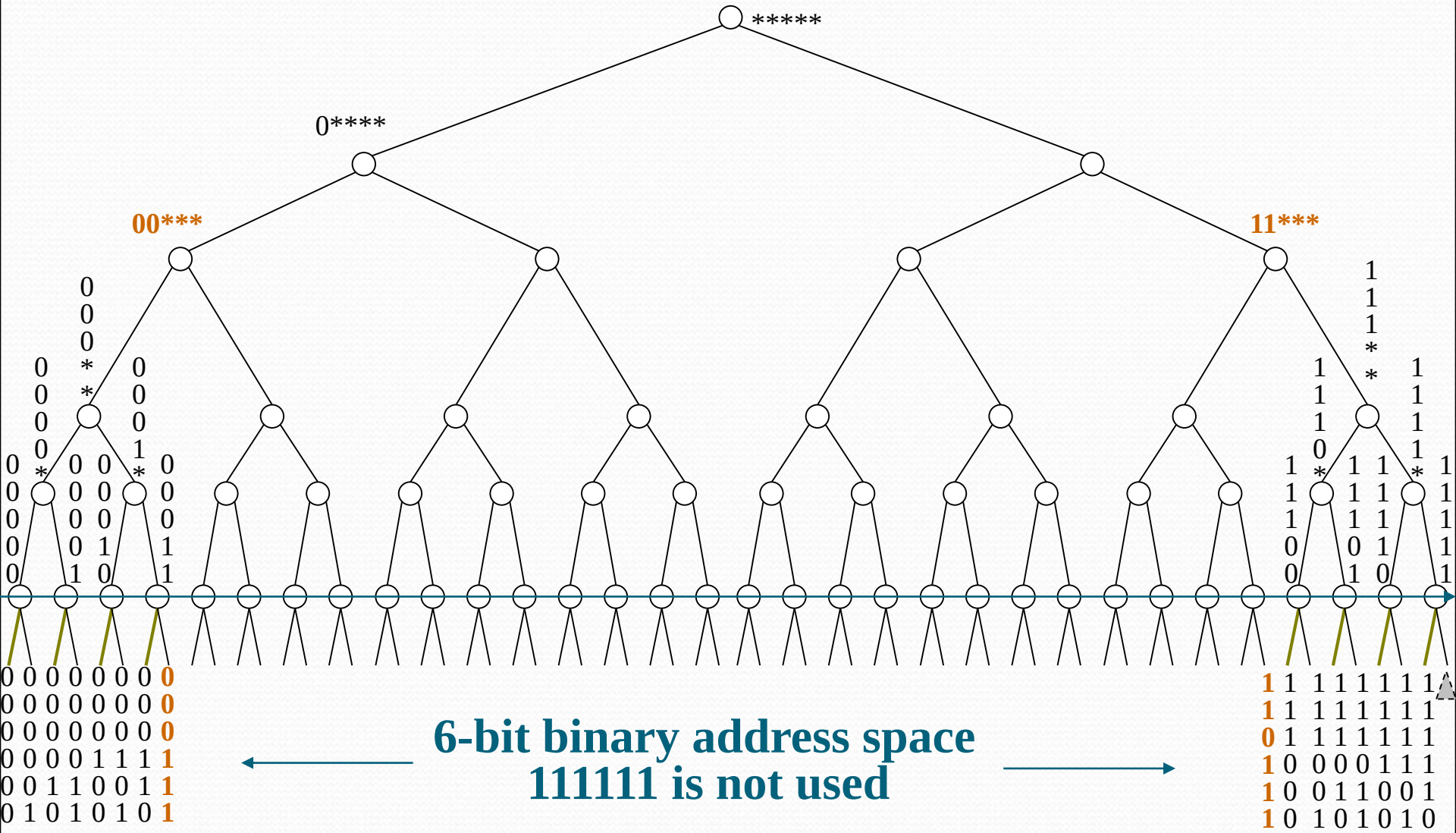
- ***(n+1)-bit* format**: $\mathbf{b_{n-1}\ldots b_{n-\ell}10\ldots0}$ **(ℓ is prefix len)**
  - for the prefix $b_{n-1}\ldots b_{n-\ell}$* of length ℓ in ternary format, there is one trailing '1' followed by n – ℓ 0's.
- or
- **(n+1)-bit format**: $\mathbf{b_{n-1}\ldots b_{n-\ell}01\ldots1}$
  - for the prefix $b_{n-1}\ldots b_{n-\ell}$* of length ℓ in ternary format, there is one trailing '0' followed by n – ℓ 1's.
- **IPv4: 32+1 bits**

5-bit Prefixes: $b_{n-1}\ldots b_{n-\ell}10\ldots0$



6-bit binary address space
000000 is not used

5-bit Prefixes: $b_{n-1}\ldots b_{n-\ell}01\ldots1$



6-bit binary address space
111111 is not used

(n+1)-bit prefix representation

- For efficient comparisons between prefixes
- (n+1)-bit representation for n-bit prefix.

  $b_{n-1}\ldots b_{n-i}*\ldots* \rightarrow b_{n-1}\ldots b_{n-i}\underline{10\ldots0}$

  with n – i trailing zeros.
- n = 32 for IPv4 or n = 64 for IPv6
- Examples: 5-bit prefixes
  - 110** → 110100
  - 10000 → 100001
- Converted n-bit representation

Prefixes in the format of Ranges

- Why ranges?
  - Prefixes can also be represented by ranges.
  - The source/destination port fields of rule tables for packet classification are ranges.
- Prefixes are special cases of ranges.
- Prefix $b_{n-1}...b_{n-\ell}*$ of length ℓ is the range of addresses
  - $[b_{n-1}\ldots b_{n-\ell}0\ldots0, b_{n-1}\ldots b_{n-\ell}1\ldots0]$
- *Overlapping*:
  - Two ranges are not disjoint.
- *Partially overlapping:*
  - *Two ranges are neither disjoint nor enclosing.*

Existing Binary Range Search

Traditional Endpoint: [e, f] → *e* and *f*.

Range	0–3	5–5	7–17	10–15	21–28	23–23
Port or ID	A	B	C	D	E	F

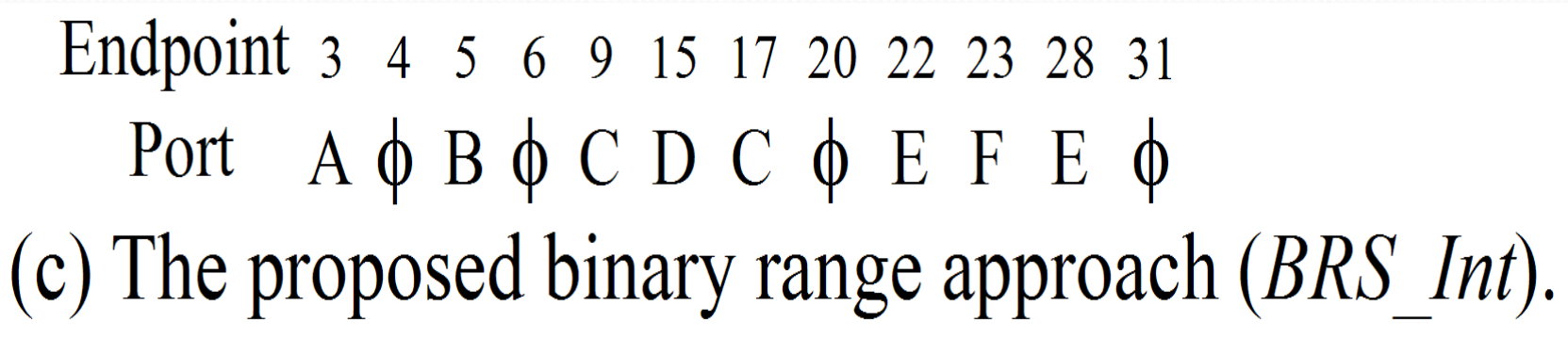
(a) The list of six 5-bit original ranges.

Endpoint	0	3	5	7	10	15	17	21	23	28	31
= Port	A	A	B	C	D	D	C	E	F	E	ϕ
> Port	A	ϕ	ϕ	C	D	C	ϕ	E	E	ϕ	-

(b) The binary range approach proposed in [14].

Proposed Binary Range Search

Proposed Endpoint: minus-1-endpoint range [e, f] → *e – 1* and *f*.

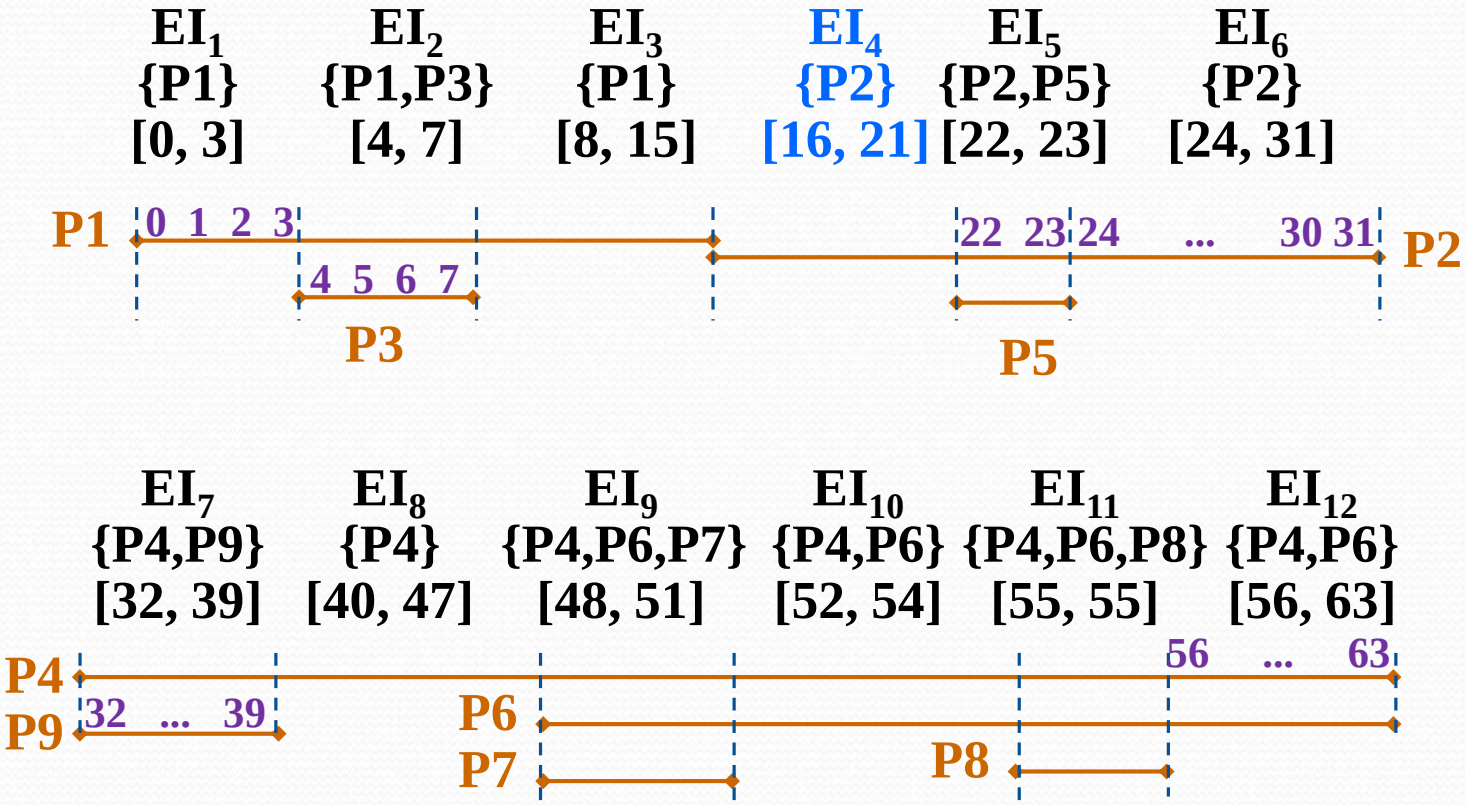


(c) The proposed binary range approach (*BRS_Int*).

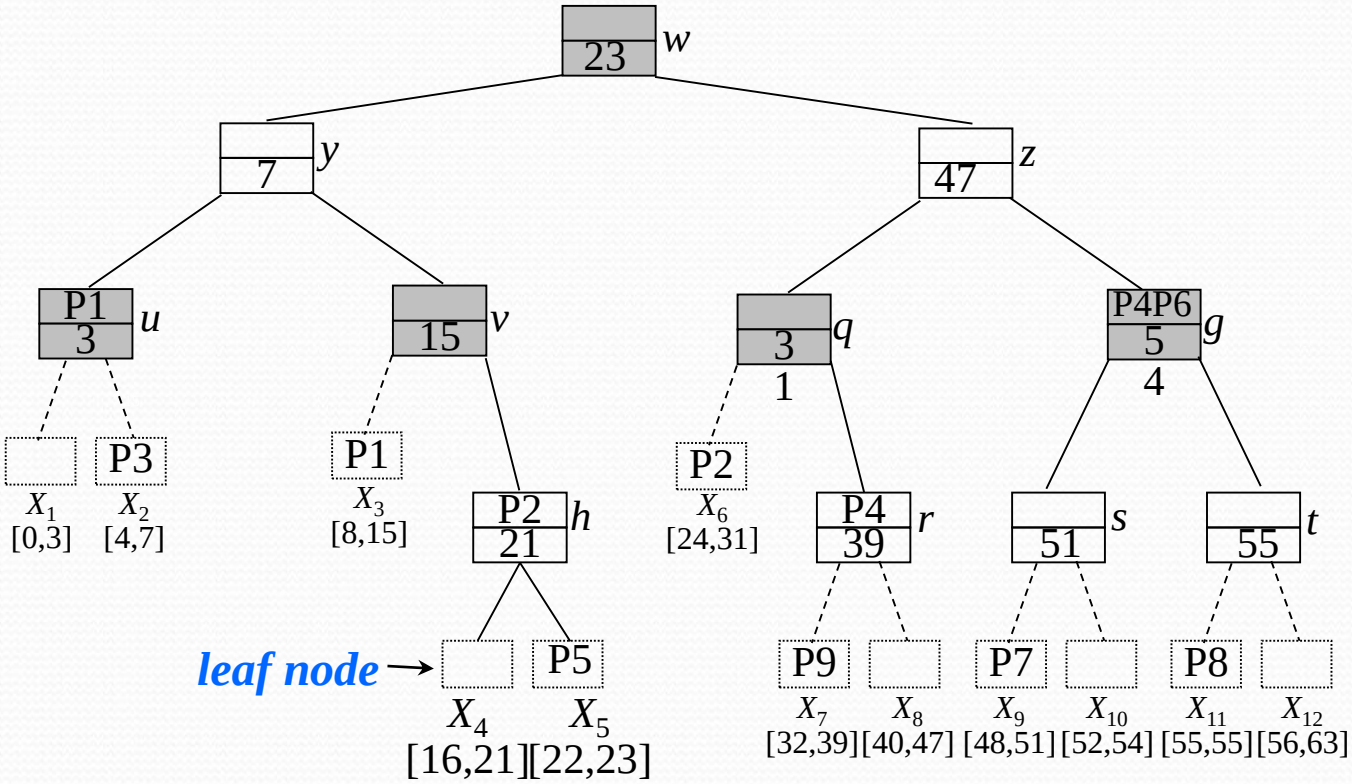
Elementary Intervals for Ranges

- Graphical view

P1 [0 , 15]
P2 [16, 31]
P3 [4 , 7]
P4 [32, 63]
P5 [22, 23]
P6 [48, 63]
P7 [48, 51]
P8 [55, 55]
P9 [32, 39]



Segment Tree

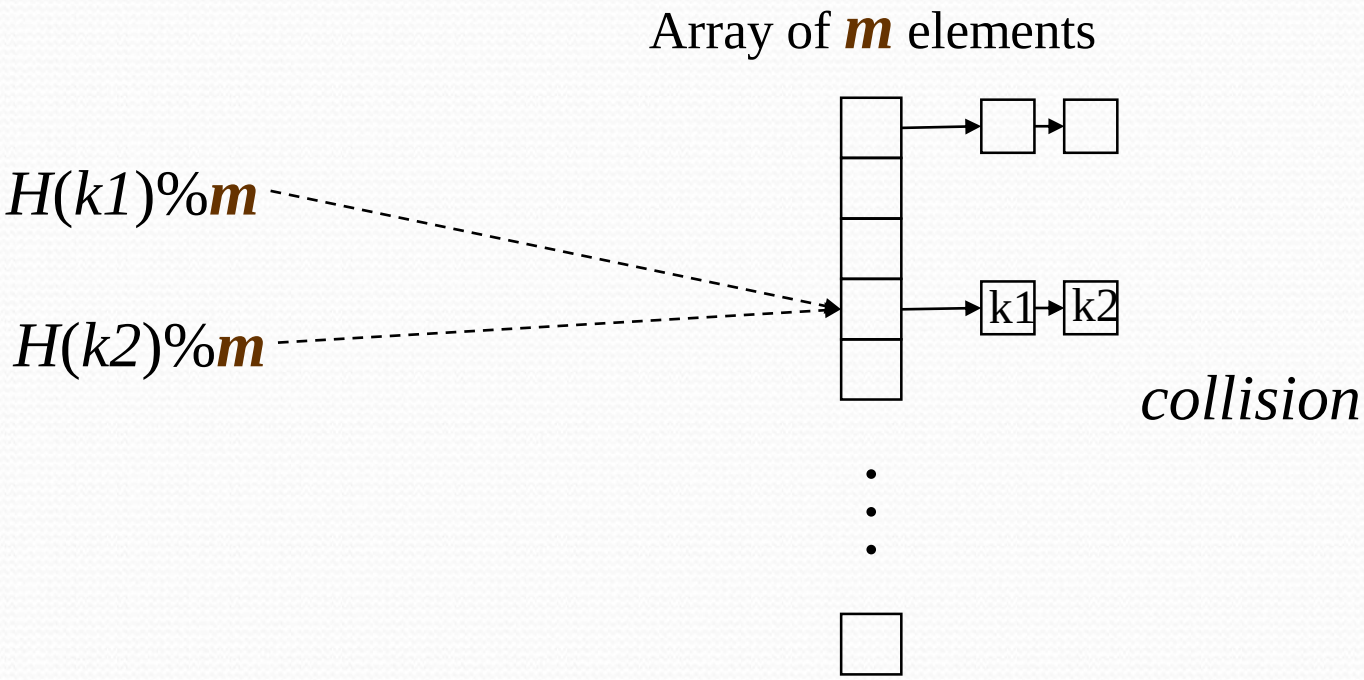


Hash Table

- Narrowing down the search space.
- Index = Hash_function(*key*)%*m*, where *key* may be the first *k* bits of IP addresses and *m* is the size of the hash table.
- Perfect hash: no collision
- Minimal perfect hash: A perfect hash, where the size of its hash table is *k* for *k* different hashing keys.

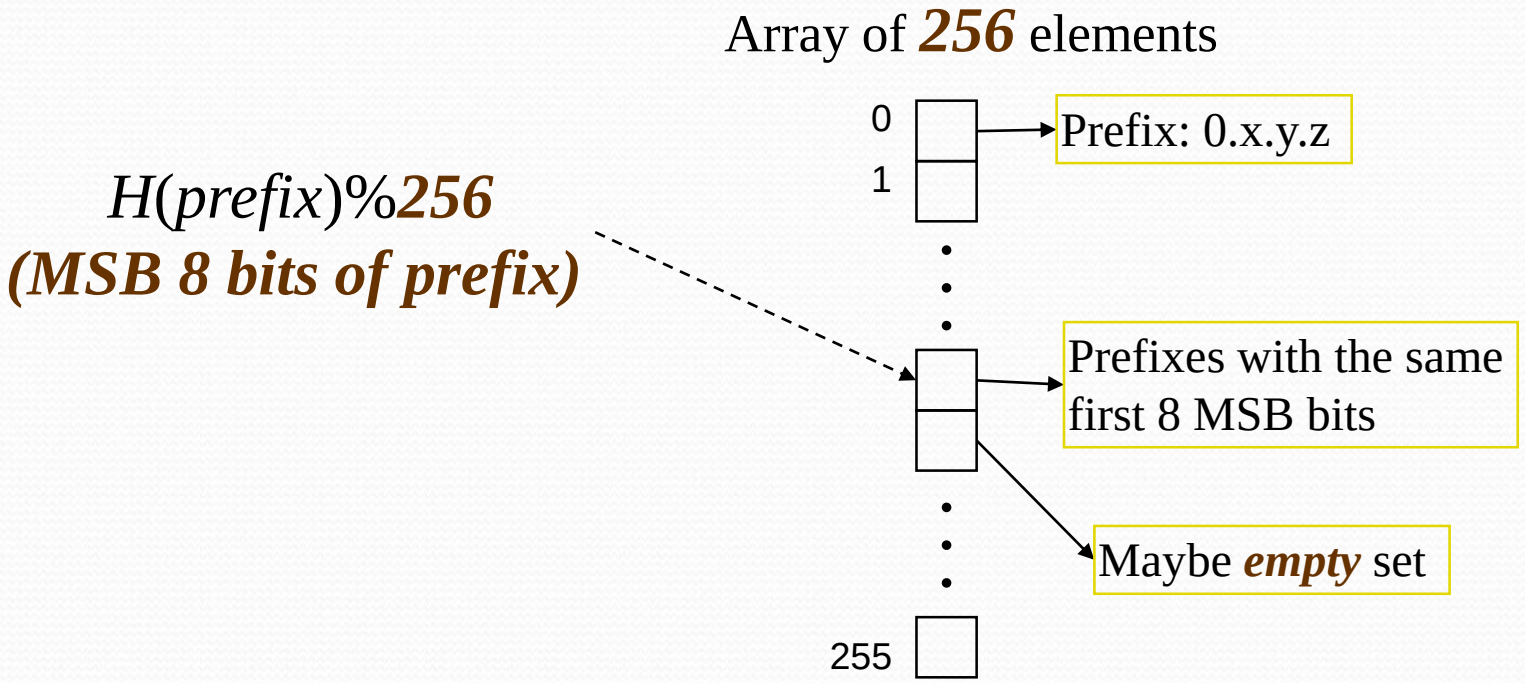
Hash Table

- Difficulties: prefixes and ranges can not be used as the keys of the hash functions directly.



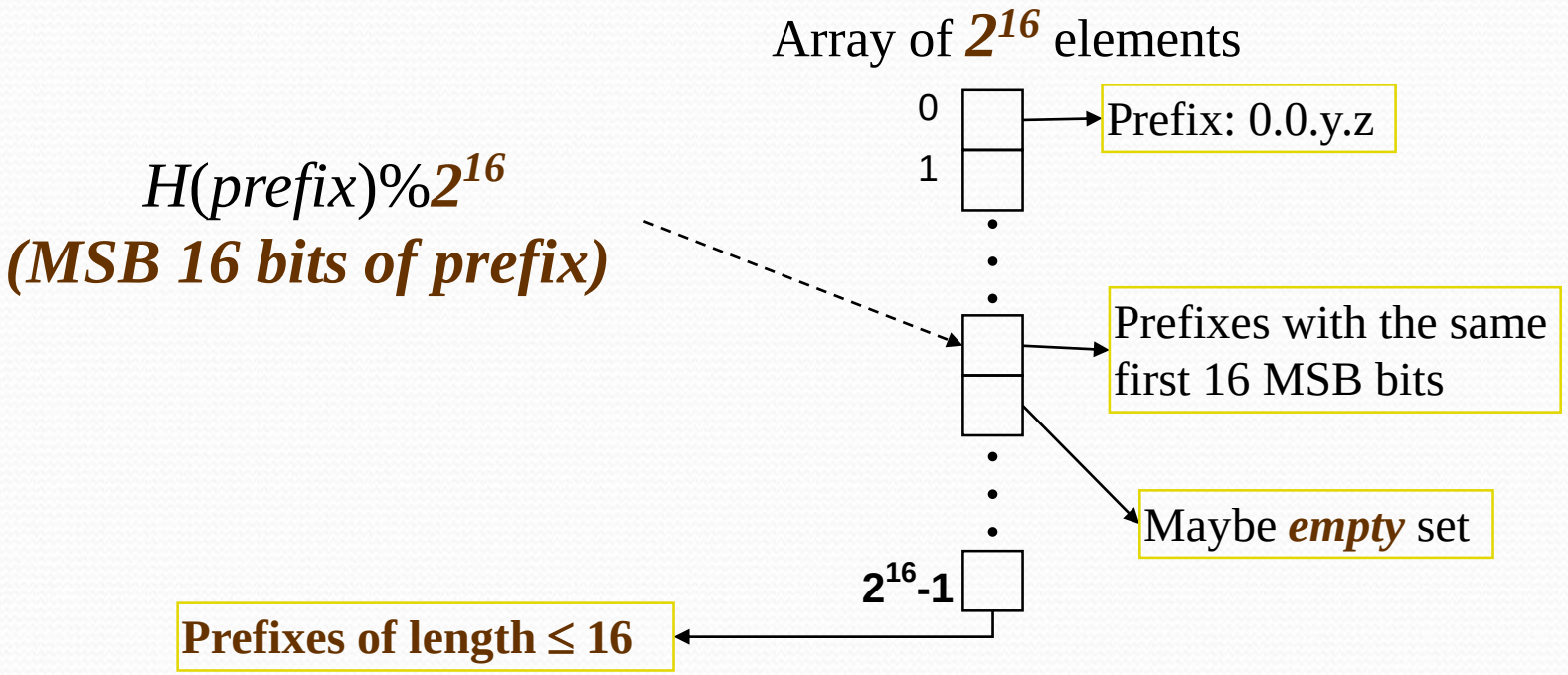
Hash Table: 8-bit Segmentation table

- A 8-bit segmentation table is usually used for IPv4 forwarding tables because there is no prefix of length shorter than 8.



Hash Table: 16-bit Segmentation table

- Prefixes of length <= 16 must be stored properly.
  - For example, duplicate 0.0.b.c/15 into buckets 0 and 1 or store the port of 0.0.b.c/15 into elements 0 and 1.
  - Put them into another set (good for update but need to search two sets in the worst case).



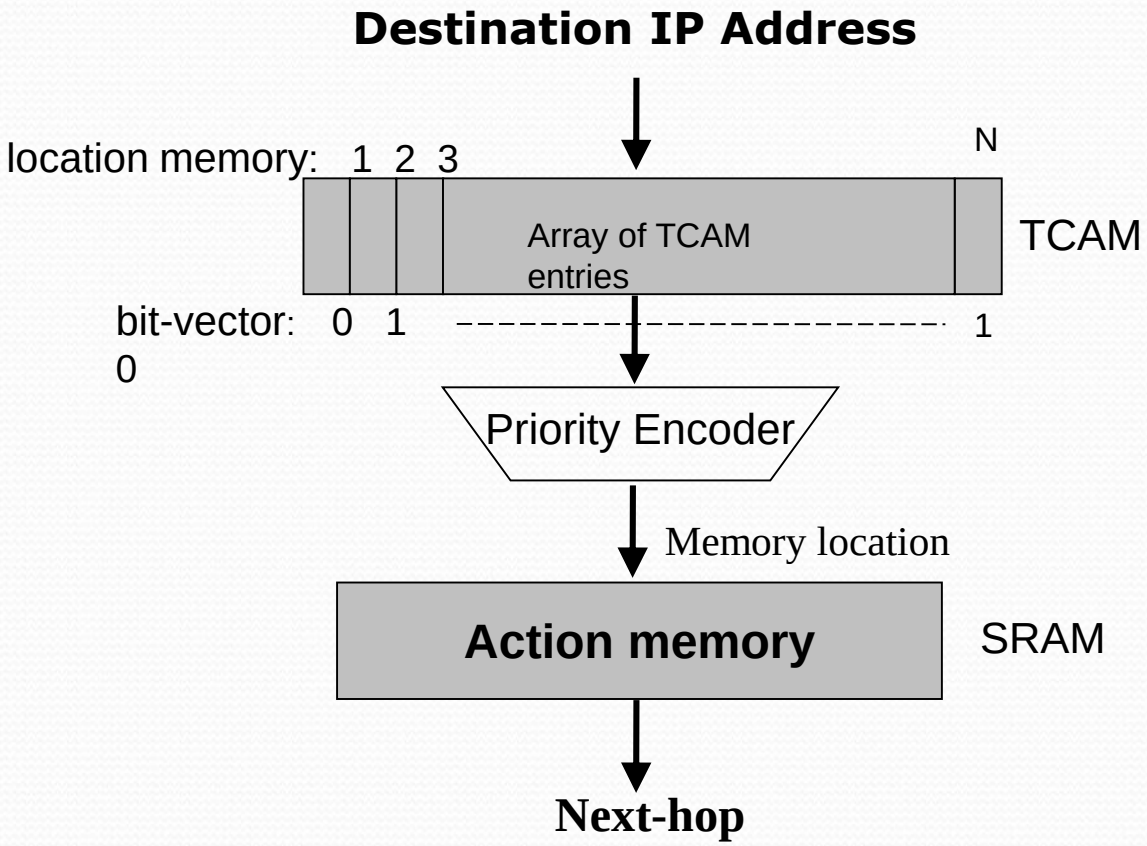
Introduction – TCAM

- Content-addressable memories(CAMs) enable a search operation to complete in a single clock cycle.
- TCAM allows a third state of "*" or "Don't Care" for adding flexibility to the search.
- The TCAM memory array stores rules in decreasing order of priorities for simplifying the priority encoder.

Introduction – TCAM

- Compare input key against all TCAM entries in parallel.
  - Each TCAM entry is a prefix for IP lookups
  - In general, each entry is a ternary string
- The *N*-bit bit-vector indicates which rules match.
- *N*-bit priority encoder indicates the address of the matched entry with highest priority. The address is used as an index into an array in SRAM to find the action associated with this prefix.

TCAM ip lookup example



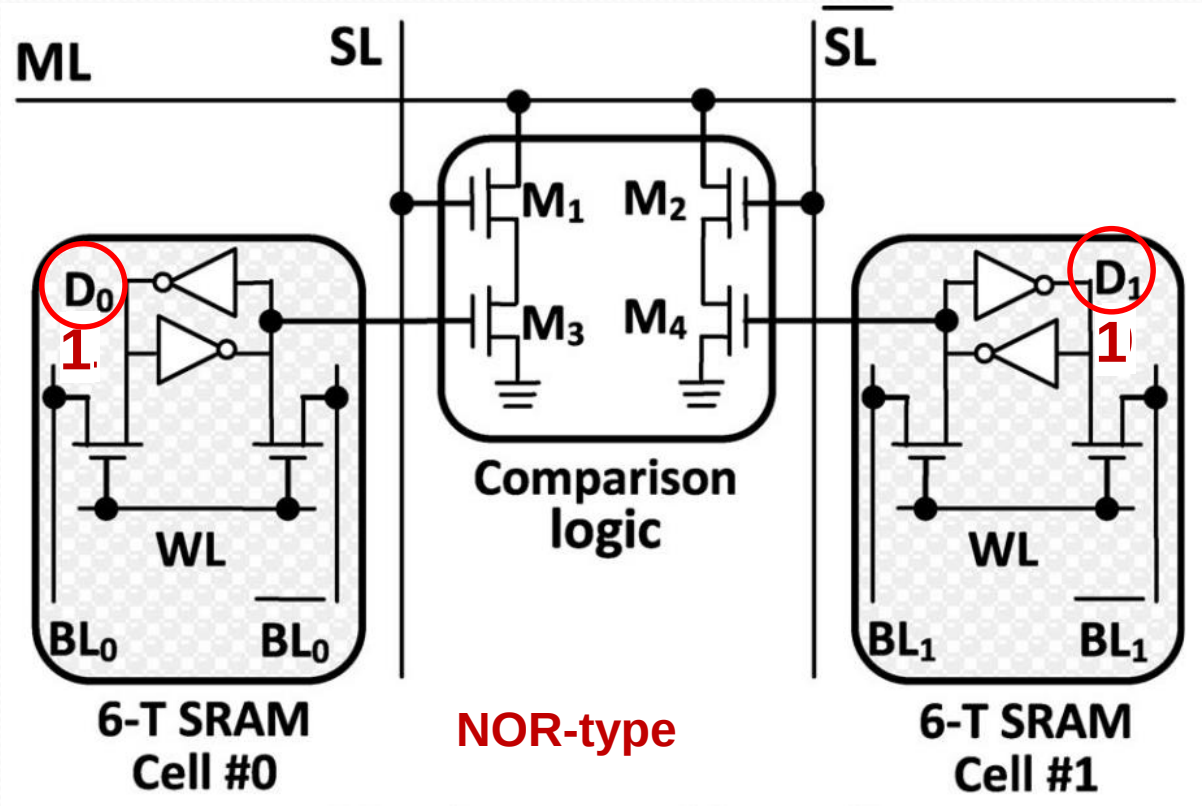
Type of CAMs

- Binary CAM (BCAM or CAM) only stores 0s and 1s
  - Applications: MAC table consultation. Layer 2 security related VPN segregation.
- Ternary CAM (TCAM) stores 0, 1 and *.
  - Application: when we need wilds cards such as, layer 3 and 4 classification for QoS and CoS purposes and IP routing (longest prefix matching).
- Available **sizes**: 1Mb, 2Mb, 4.7Mb, 9.4Mb, and 18.8Mb, 20Mb, 36 Mb, 40Mb.
- **Speed**: 50, 100, 360MPP
- CAM entries are structured as multiples of 36-40 bits rather than 32 bits.

TCAM cell

- each cell in one of three logic states: 0,1, or *.
- two 1-bit 6-T static random-access memory (SRAM) cells (D_0/D_1) are used to store three logic states of a TCAM cell.

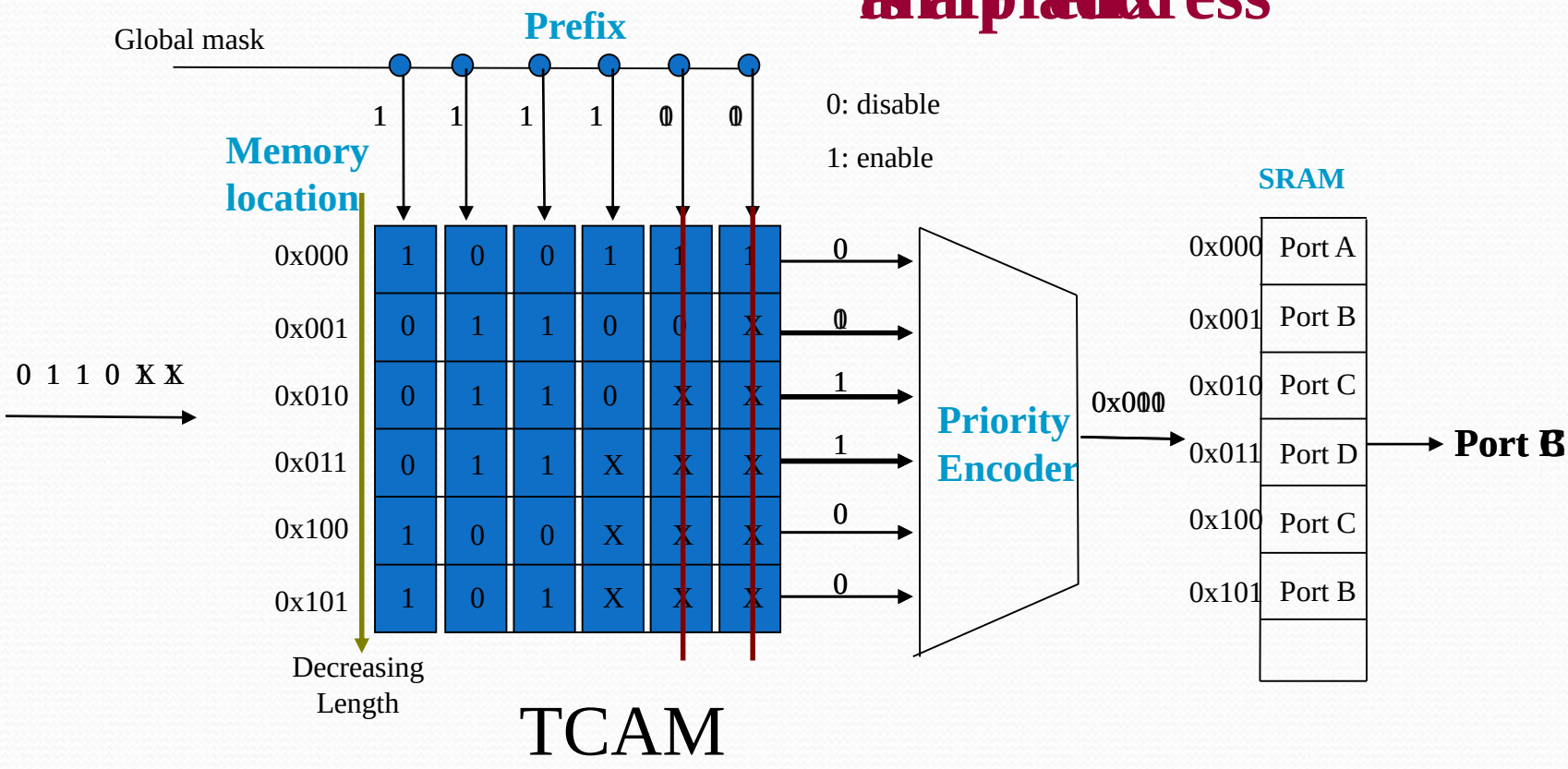
D=*.



TCAM

- **Longest prefix match using TCAM**

Case : Search key is in all Prefix address



Binary prefix search

- Definition 1 (Prefix comparison):

  The inequality **0 < * < 1** is used to compare two prefixes in the ternary format.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Port	G	E	H	L	K	O	F	B	A	N	I	M	J	D	C
Prefix	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
	0	0	0	1	1	1	1	1	0	0	0	0	0	0	1
	0	0	1	0	0	0	0	*	*	1	1	1	1	1	0
	1	1	1	0	0	1	1			1	1	1	1	1	*
	0	*	0	1	1	1	1			0	0	0	0	*	
	*		0	1	1	0	*			0	0	0	1		
			*	0	0	0				0	1	1	0		
				0	*	1				1	*	1	*		

Fig. 1. List of sorted prefixes based on the rule of 0 < ∗ < 1.

Binary prefix search

- Directly performing a binary search on the list of sorted prefixes may encounter a failure: Dst = **01011000**

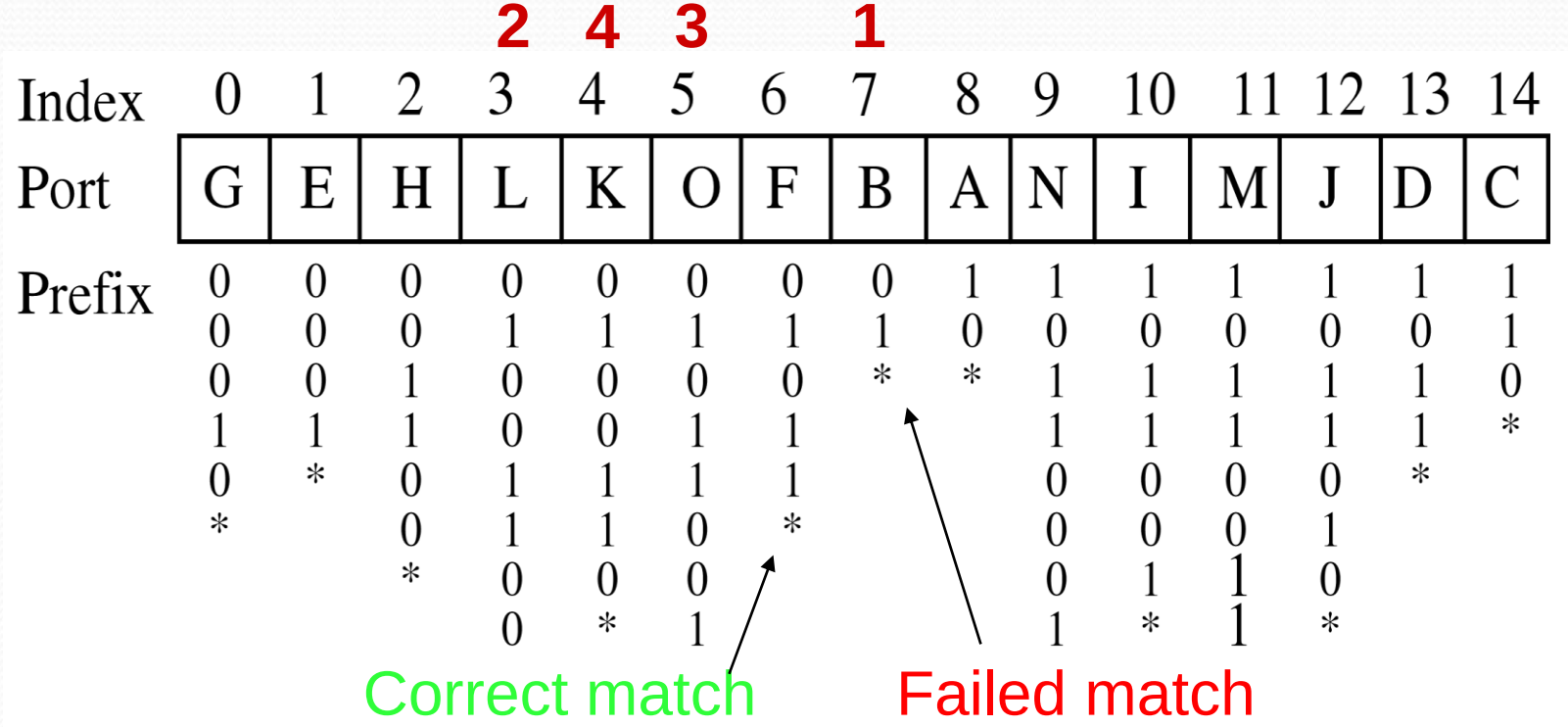


Fig. 1. List of sorted prefixes based on the rule of $0 < * < 1$.

Binary prefix search

- Enclosure relationship between prefixes results in the search failure
- Generate some **auxiliary** prefixes that inherit the routing information of the original LPM (e.g., F) and put them where the binary search operations can find them. ex. ***auxiliary prefix 01011000***.
- Therefore, it is feasible to split prefix F into two parts such that both sides of prefix O are covered.

Binary prefix search

- The full tree expansion.

  Split enclosure prefixes into many longer prefixes (**leaf pushing**).

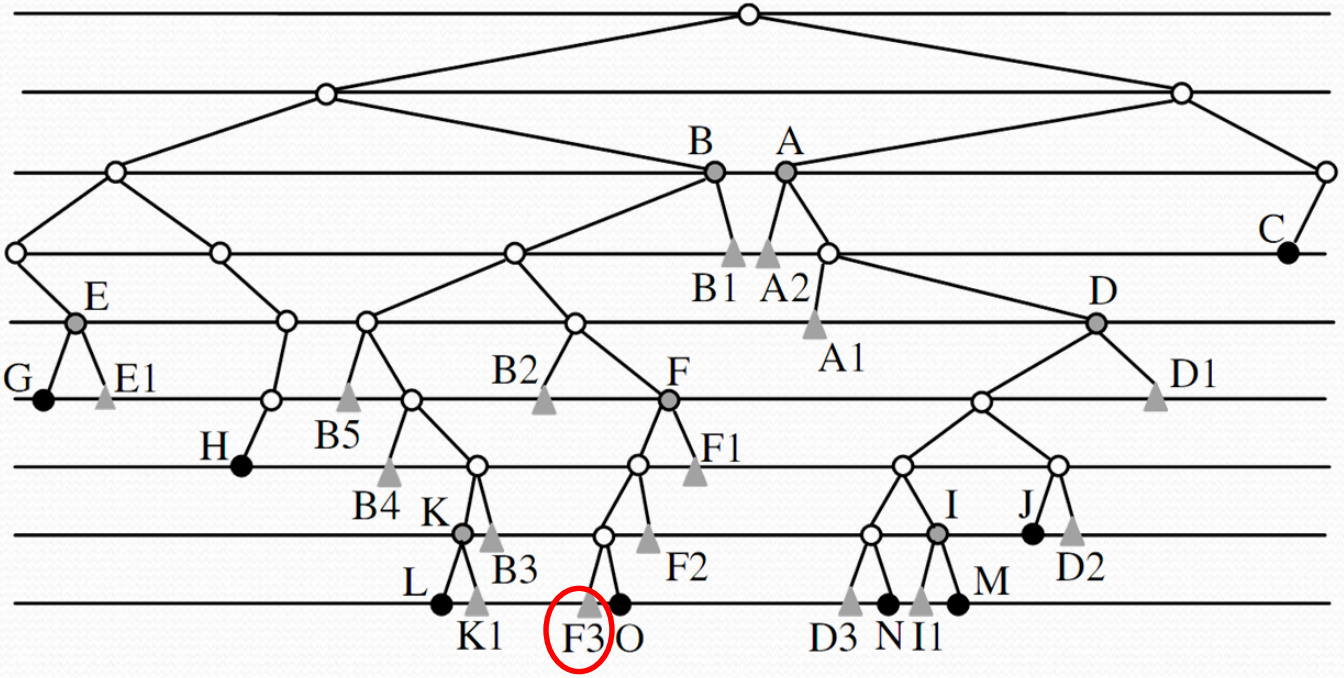
- Auxiliary prefix merge

  Many auxiliary prefixes may inherit the same routing information of a common enclosure prefix. These prefixes can be merged into one. The merge operation is defined as follows.

  - The longest common ancestor (LCA) of a set of consecutive prefixes in the binary trie.

Binary prefix search

- The full tree expansion



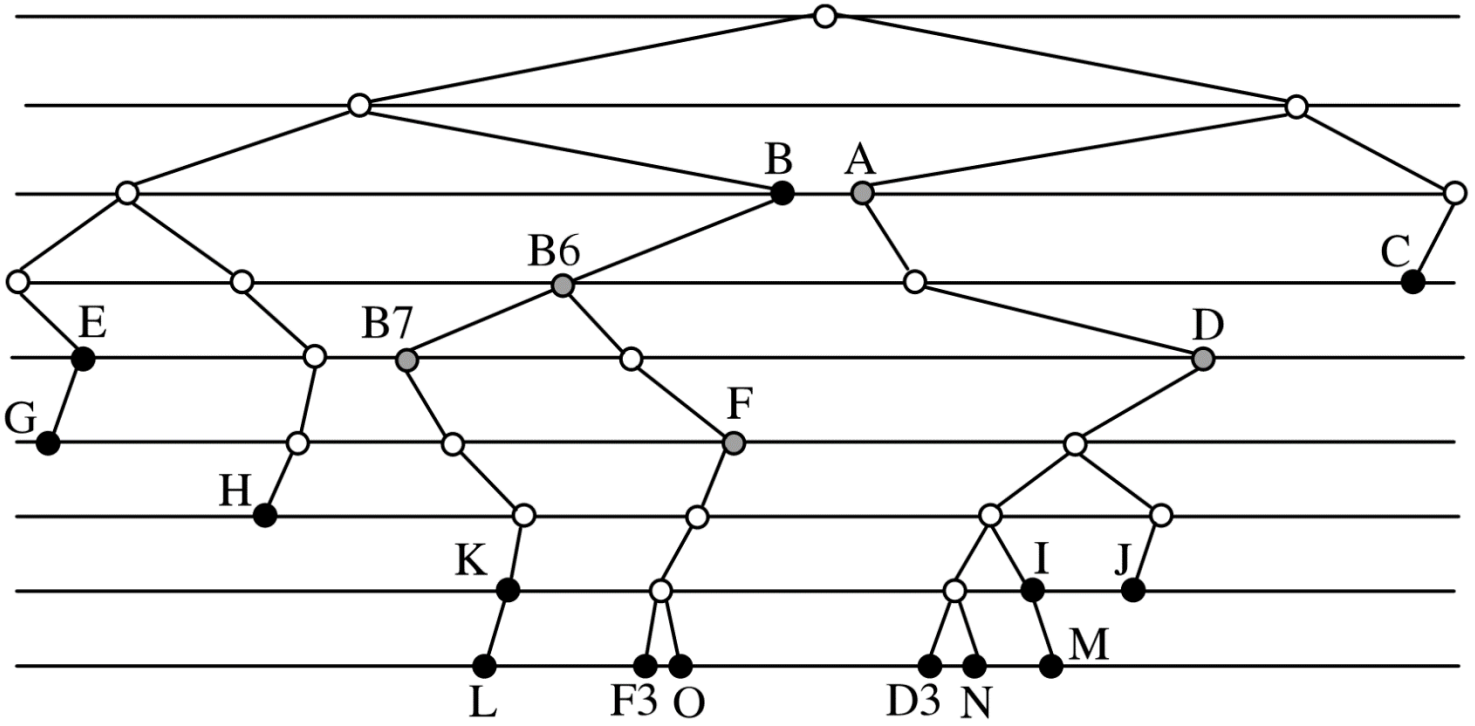
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
G	E1	H	B5	B4	L	K1	B3	B2	F3	O	F2	F1	B1	A2	A1	D3	N	I1	M	J	D2	D1	C

F3=01011000

Fig. 2. The full tree expanded from the binary trie for prefixes in Fig. 1.

Binary prefix search

- The full tree after the merge operations

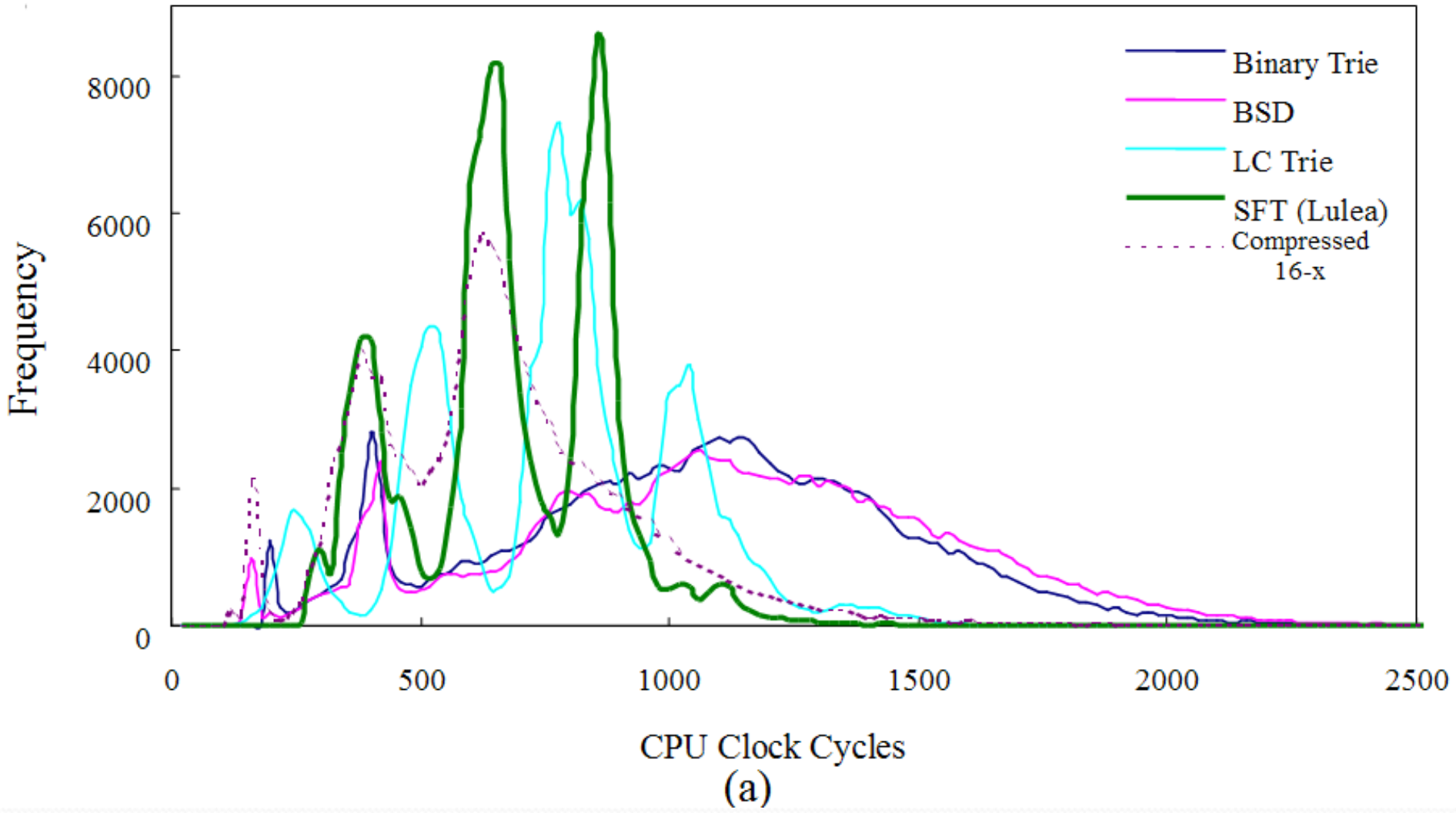


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
G	E	H	B7	L	K	B6	F3	O	F	B	A	D3	N	I	M	J	D	C

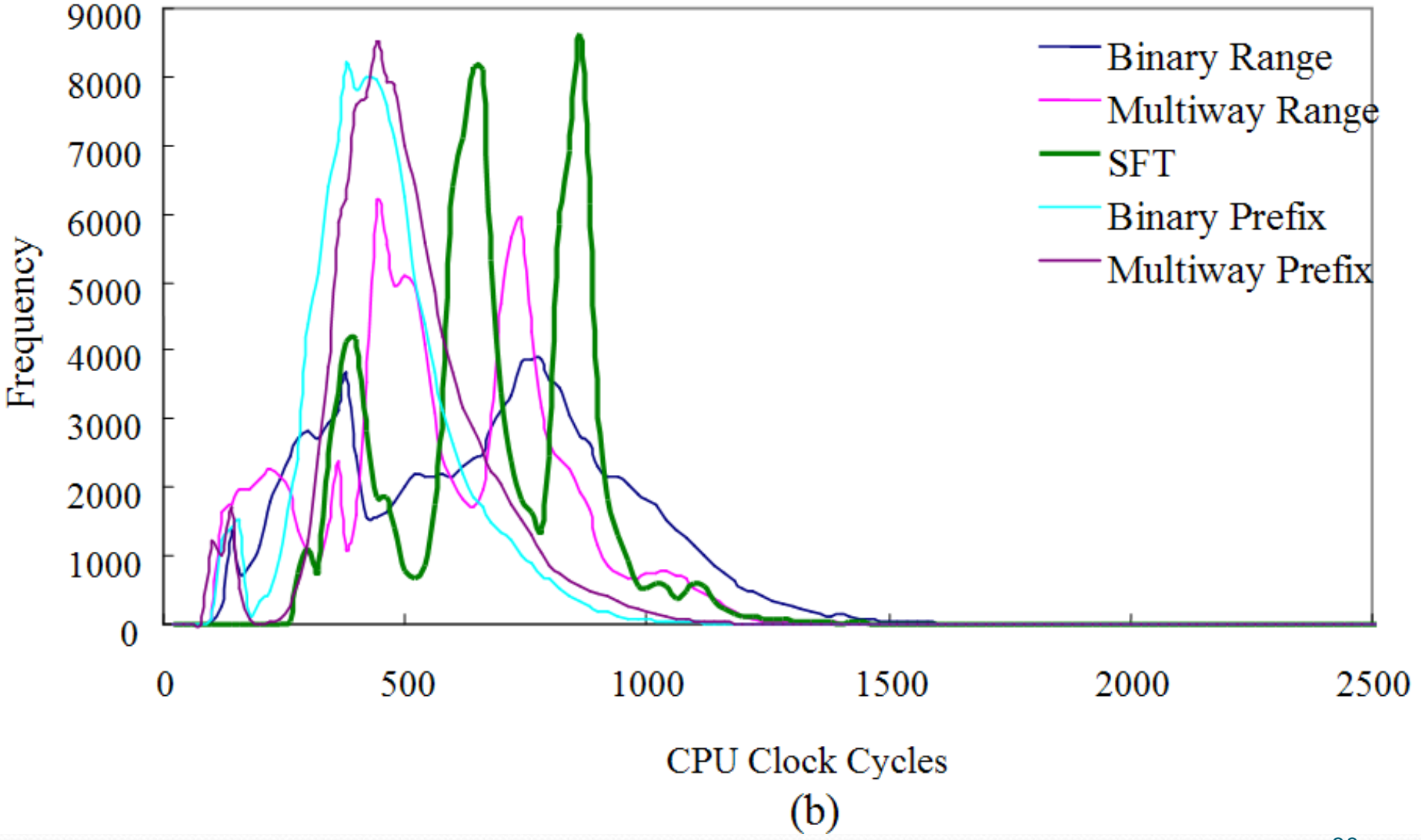
F3=01011000

Fig. 3. The tree after merge operations.

Performance: Search Speed



Performance: Search Speed



(b)

Scheme	Segmentation	Statistics	Ref.	Memory
Original table	N/A	# of prefixes: 120,635 (45 bits each)	N/A	662.7 KB
Binary trie	16-bit segmentation 65535 entries (4 byte each)	# of nodes: 320,478 (7 bytes each)	8/32	2,447 KB
BSD trie		# of nodes: 222,334 (8 bytes each)	8/26	1,993 KB
Compressed 16-x		Base array: 427 KB Compressed Bit-map: 427 KB CNHA: 37.9 KB	1/3	1,147 KB
LC trie	Branch factor: 16 Fill factor: 0.5	# of nodes: 259,371 (4 bytes each) Base vector: 110,679 (16 bytes each) Prefix vector: 9,927 (12 bytes each) Next Hop vector: 255 (4 bytes each)	1/5	2,859 KB
SFT	level-1 pointers: 13,317 level-2 pointers: 461 (4 bytes each)	# of segments (avg # of prefixes per segment) Sparse: 2,765 (2.9) Dense: 4,300 (25.5) Very dense: 586 (91.7) Maptable: 5.3K Base array: 2K Code Word array: 8K	1/12	649.9 KB
Binary range	No segmentation	# of prefixes: 232,887 (6 bytes each)	1/6	1,365 KB
Binary range	16-bit segmentation	# of prefixes: 217,146 (4 bytes each)	1/4	1,104 KB
Multiway range	16-bit segmentation	# of blocks: 71,034 (64 bytes each)	1/4	4,695 KB
Binary prefix	No segmentation	# of prefixes: 145,737 (5 bytes each)	1/5	646 KB
Binary prefix	16-bit segmentation	# of prefixes: 117,968 (3 bytes each)	1/4	601 KB
Multiway prefix	16-bit segmentation	# of blocks: 11,175 (64 bytes each)	1/3	954.4KB

Table 3: Memory required for the routing table containing 120,635 prefixes, where multiway assumes a cache block of size 64 bytes.

Metrics for Lookup Algorithms

- High Speed (ex. 40 Gbps/40-byte=128m packets/sec)
- Small storage (ex. Cache or On-Chip memory)
- Low update time
- Ability to handle large routing tables
- Flexibility in implementation
- Low preprocessing time
- IPv6

Survey: IP Lookups

- M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," IEEE Network, vol. 15, pp. 8–23, March 2001.
- **Schemes for optimizing search speed**
  - Multibit tries
  - Two-level multibit trie, 16-16, 24-8
  - Binary range search (endpoint)
  - Binary search on prefix length
  - **Binary prefix search**
  - **Binomial Spanning tries based on Hamming and Golay perfect codes**
  - **FPGA pipelined implementation (over 100Gbps)**

Existing IP lookup schemes

- **Schemes for optimizing memory requirement**
  - Small forwarding table (SFT): compressed 16-8-8 trie
  - Level compressed (LC) trie
  - Huang 's compressed 16-x (C-16-x)
  - **Compressed 8-8-8-8 trie using minimal perfect hashing function**
  - Hierarchical endpiont tree (01**→0100 and 1000)
  - Tree bitmap (compressed 4-4-4-4-4-4-4-4 trie)
  - Memory optimized multibit tries with dynamic programming

Existing IP lookup schemes

- **Schemes for optimizing update speed** (log N)

Binary

- Binary tree on binary tree scheme (PBOB),
- Priority search tree scheme (PST),
- Collection of red-black tree schemes (CRBT)
- Most Specific Prefix Tree (MSPT)
- Multigroup Most Specific Prefix Tree (MG-MSPT)
- Dynamic segment tree (DST), extending binary range search

Multiway

- Multiway range tree (MRT)
- Prefix in B-Tree (PIBT)
- Dynamic Multiway Segment Tree (DMST)

Existing IP lookup schemes

- **Schemes for optimizing IPv6**
  - Not really
  - Some dual stack (IPv4/IPv6) papers
  - 128-bit IPv6 addresses
  - 32-bit vs. 64-bit CPUs or Memory bandwidth
  - Initial results: binary search based schemes are better than trie based schemes

Goal of hardware approaches

- Pipeline architecture to achieve the throughput of over 100Gbps
- The most popular design is based on multibit trie
- Advantages
  - Simple
  - Bubble instruction to perform updates
- Disadvantages
  - Unbalanced memory among stages (solvable)
  - Memory requirement is too large to fit in the on-chip memory of hardware device such as FPGA

Packet Classification Problem

- performed by routers to classify the incoming packets into *flows* according to a set of *rules*
- If a packet matches multiple rules, the matching rule with the highest priority is returned.
- Typically, each rule contains five fields,
  - **Source IP address**,
  - **destination IP address**,
  - **source port number**,
  - **destination port number**,
  - **protocol type**

A small classifier in five dimensions

Rule	Dst. IP	Src. IP	Dst port	Src port	Protocol #	Action
R1	140.116.246.69/32	140.116.82.11/32	*	*	*	Deny
R2	140.126.53.0/24	140.116.100.157/32	80	*	tcp	Deny
R3	140.116.247.4/32	140.116.160.0/24	> 1023	*	udp	Permit
R4	0.0.0.0/0	0.0.0.0/0	*	*	*	Permit

prefix

range

Classification examples

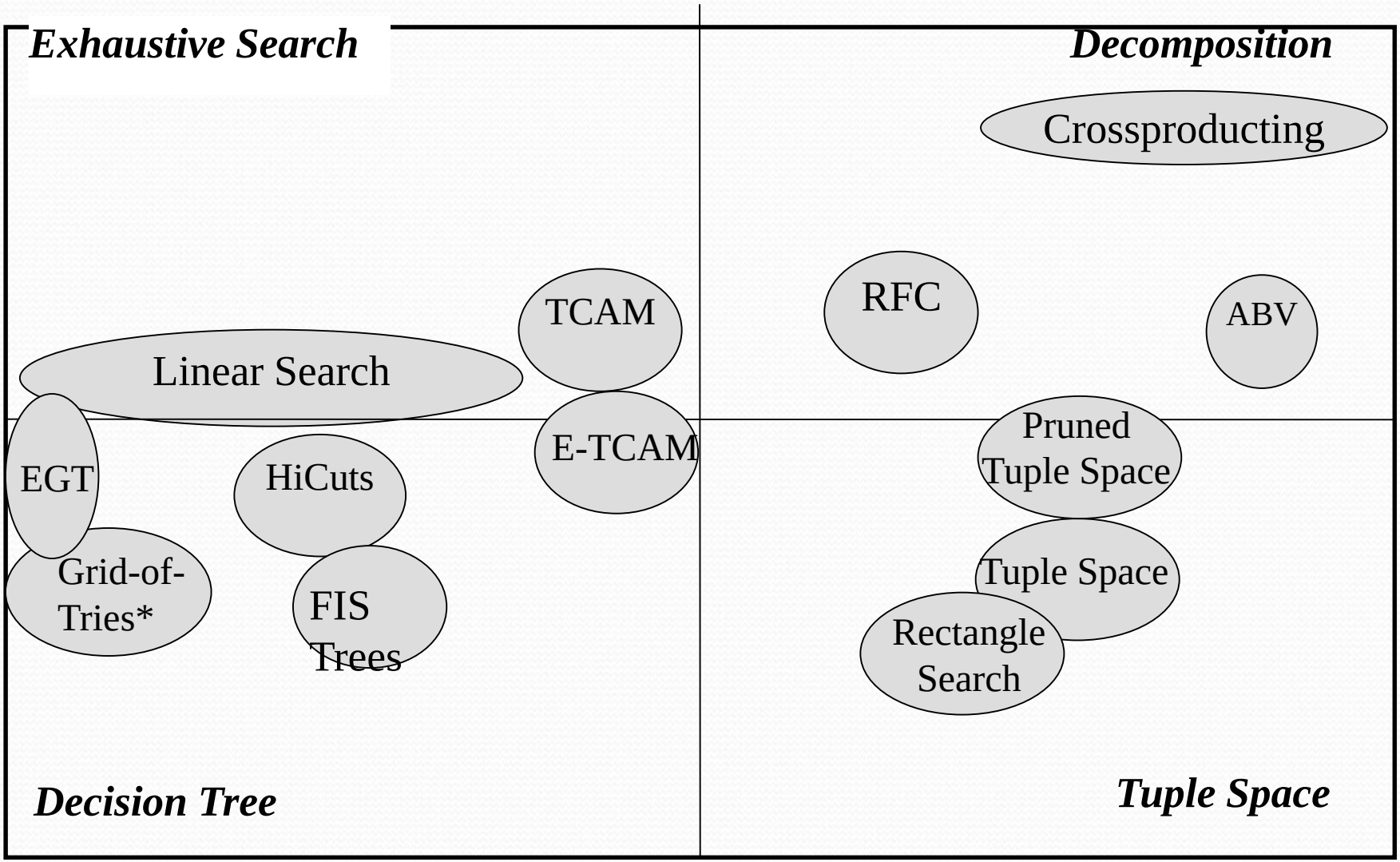
PKT	Dst. IP	Src. IP	Dst port	Src port	Protoc #	Action
P1	140.116.246.69	140.116.82.11	80	1222	tcp	Deny,R1
P2	140.126.53.10	140.116.100.57	80	1233	udp	Deny,R2
P3	140.116.247.4	140.116.160.10	1024	1235	tcp	Permit,R3

A real-life classifier in five dimensions

1	@133.201.236.126/32	68.153.25.214/32	0 : 65535	20 : 20	0x06/0xFF	0x1000/0x1000
2	@133.200.221.190/32	68.153.25.214/32	0 : 65535	1707 : 1707	0x06/0xFF	0x1000/0x1000
3	@133.205.191.235/32	68.153.25.214/32	0 : 65535	65500 : 65500	0x06/0xFF	0x0000/0x0200
4	@133.204.255.123/32	68.153.25.214/32	0 : 65535	6789 : 6789	0x06/0xFF	0x0000/0x0200
5	@133.207.151.50/32	68.153.25.214/32	0 : 65535	1711 : 1711	0x06/0xFF	0x1000/0x1000
6	@133.194.174.193/32	68.153.25.214/32	0 : 65535	5632 : 5632	0x06/0xFF	0x1000/0x1000
7	@133.193.121.141/32	68.153.25.214/32	0 : 65535	21 : 21	0x06/0xFF	0x0000/0x0200
8	@133.193.143.215/32	174.33.117.29/32	0 : 65535	27400 : 27400	0x06/0xFF	0x0000/0x0200
9	@133.193.204.3/32	68.153.25.214/32	0 : 65535	1221 : 1221	0x06/0xFF	0x1000/0x1000
10	@133.198.140.156/32	253.175.13.67/32	0 : 65535	1707 : 1707	0x06/0xFF	0x0000/0x0200
11	@133.219.96.219/32	68.153.25.214/32	0 : 65535	1600 : 1600	0x06/0xFF	0x1000/0x1000
12	@133.217.155.69/32	68.153.25.214/32	0 : 65535	5632 : 5632	0x06/0xFF	0x1000/0x1000
13	@133.223.245.25/32	68.153.25.214/32	0 : 65535	1221 : 1221	0x06/0xFF	0x0000/0x0200
14	@133.223.184.25/32	68.153.25.214/32	0 : 65535	1733 : 1733	0x06/0xFF	0x0000/0x0200
15	@133.208.167.157/32	68.153.25.214/32	0 : 65535	1521 : 1521	0x06/0xFF	0x0000/0x0200
16	@133.213.135.135/32	68.153.25.214/32	0 : 65535	1704 : 1704	0x06/0xFF	0x1000/0x1000
17	@133.227.82.6/32	68.153.25.214/32	0 : 65535	1521 : 1521	0x06/0xFF	0x1000/0x1000
18	@133.227.109.2/32	174.33.117.29/32	0 : 65535	32201 : 32201	0x06/0xFF	0x1000/0x1000
19	@133.225.150.22/32	68.153.25.214/32	0 : 65535	1489 : 1489	0x06/0xFF	0x1000/0x1000
20	@133.224.164.116/32	68.153.25.214/32	0 : 65535	5632 : 5632	0x06/0xFF	0x1000/0x1000
21	@133.231.250.178/32	174.33.117.29/32	0 : 65535	1733 : 1733	0x06/0xFF	0x0000/0x0200
22	@133.232.114.250/32	174.33.117.29/32	0 : 65535	1706 : 1706	0x06/0xFF	0x0000/0x0200
23	@133.238.206.225/32	68.153.25.214/32	0 : 65535	1708 : 1708	0x06/0xFF	0x0000/0x0200
24	@133.237.157.174/32	68.153.25.214/32	0 : 65535	1521 : 1521	0x06/0xFF	0x0000/0x0200
25	@133.236.243.19/32	253.175.13.67/32	0 : 65535	5555 : 5555	0x06/0xFF	0x1000/0x1000
26	@133.254.103.186/32	68.153.25.214/32	0 : 65535	30211 : 30211	0x06/0xFF	0x0000/0x0200
27	@133.253.99.86/32	68.153.25.214/32	0 : 65535	6790 : 6790	0x06/0xFF	0x0000/0x0200
28	@133.244.196.230/32	68.153.25.214/32	0 : 65535	6849 : 6849	0x06/0xFF	0x1000/0x1000
29	@133.244.79.159/32	68.153.25.214/32	0 : 65535	5541 : 5541	0x06/0xFF	0x1000/0x1000
30	@133.145.5.193/32	253.175.13.67/32	0 : 65535	1708 : 1708	0x06/0xFF	0x0000/0x0200
31	@133.150.211.2/32	253.175.13.67/32	0 : 65535	1521 : 1521	0x06/0xFF	0x1000/0x1000
32	@133.155.243.148/32	68.153.25.214/32	0 : 65535	1708 : 1708	0x06/0xFF	0x1000/0x1000
33	@133.155.59.140/32	68.153.25.214/32	0 : 65535	1526 : 1526	0x06/0xFF	0x0000/0x0200
34	@133.155.32.152/32	68.153.25.214/32	0 : 65535	5540 : 5540	0x06/0xFF	0x1000/0x1000
35	@133.152.178.39/32	68.153.25.214/32	0 : 65535	6849 : 6849	0x06/0xFF	0x1000/0x1000
36	@133.152.252.112/32	253.175.13.67/32	0 : 65535	1521 : 1521	0x06/0xFF	0x0000/0x0200
37	@133.153.87.24/32	68.153.25.214/32	0 : 65535	1706 : 1706	0x06/0xFF	0x1000/0x1000

Normal text file length : 8,061,348 lines : 99,340 Ln : 1

David Taylor's Survey



Packet Classification

- **David E. Taylor**, Survey & Taxonomy of Packet Classification Techniques, **ACM Computing Surveys**, Volume 37, Issue 3, 238-275, September **2005**.
- Hierarchical trie, set-prunning trie, grid of trie
- Hierarchical binary search structures
- Hierarchical, set-prunning, grid of segment tree
- 2-phase schemes, 5 independent 1-D search+merge
  - Lucent Bit vector
  - Cross-product + Bloom Filter
  - TCAM range encoding
- **FPGA pipelined implementation (over 50Gbps)**
  - **USC Prashana's group**

Packet Classification solution

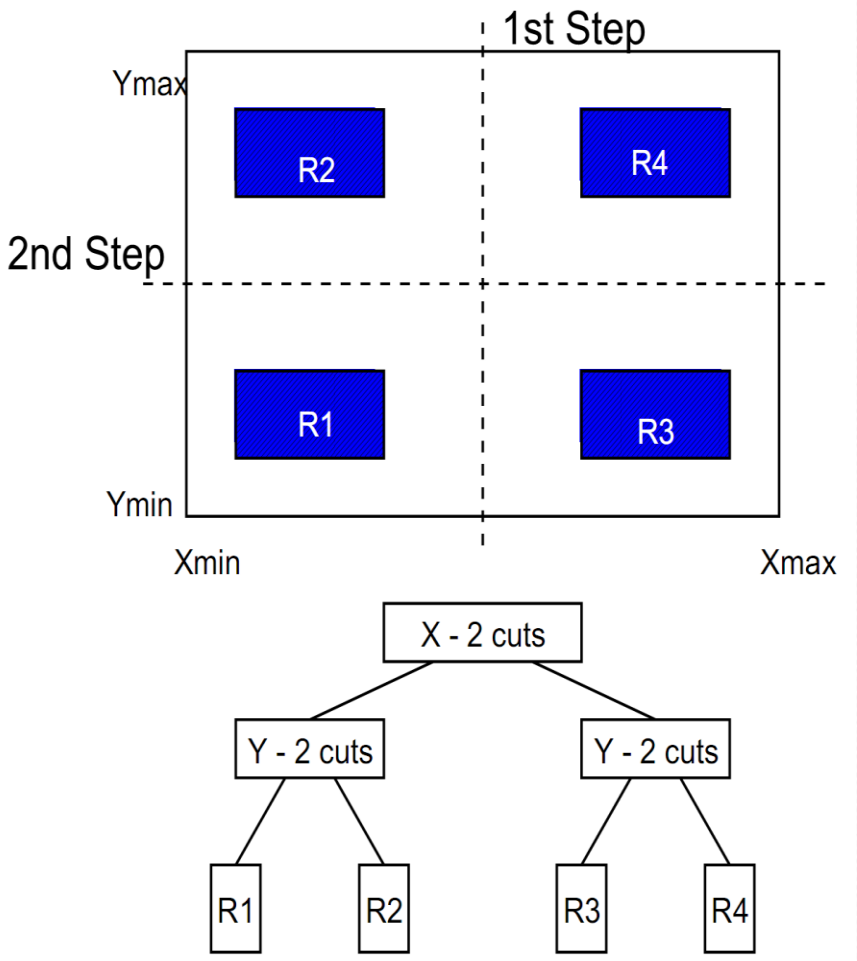
- **Hardware** (ASIC and FPGA) are usually adopted because of extremely high classification rate, but difficult to upgrade to other platforms to support new applications or consume too much electric power and board area for large classifiers
- **TCAM**: Ternary Content Addressable Memory
- **Network processor** is a programmable processor designed for network applications with the advantages of high performance of ASIC and the programming flexibility.
- **SRAM**: Algorithm + Data structure

HiCuts vs HyperCuts

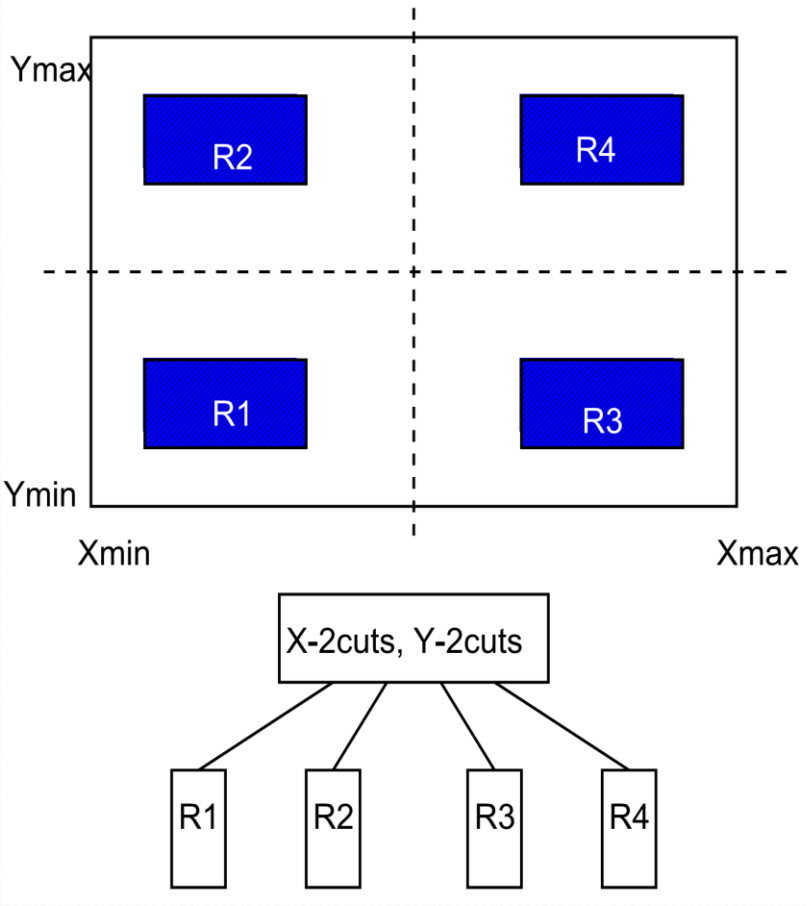
- **HiCuts** builds a **decision tree** using *local optimization* decisions at each node to choose the next dimension to test how many cuts to make in the chosen dimension. The leaves of HiCuts tree store a list of rules that may match the search path to the leaf.
- **HyperCuts** is also a **decision tree** structure, in which many dimensions are considered altogether instead of one dimension at a time in Hicuts, in order to reduce the height of decision tree.

HiCuts vs HyperCuts(con.)

HiCuts



HyperCuts



HiCuts Example

Rule	$Field_1$	$Field_2$	$Field_3$	$Field_4$	$Field_5$	ACTION
R_0	000∗	111∗	10	∗	UDP	act_0
R_1	000∗	111∗	01	10	UDP	act_0
R_2	000∗	10∗	∗	10	TCP	act_1
R_3	000∗	10∗	∗	01	TCP	act_2
R_4	000∗	10∗	10	11	TCP	act_1
R_5	0∗	111∗	10	01	UDP	act_0
R_6	0∗	111∗	10	10	UDP	act_0
R_7	0∗	1∗	∗	∗	TCP	act_2
R_8	∗	01∗	∗	∗	TCP	act_2
R_9	∗	0∗	∗	01	UDP	act_0
R_{10}	∗	∗	∗	∗	UDP	act_3
R_{11}	∗	∗	∗	∗	TCP	act_4

Figure 2: A simple example with 12 rules on five fields.

Rule	$Field_1$	$Field_2$	$Field_3$	$Field_4$	$Field_5$	ACTION
R_0	0 – 1	14 – 15	2	0 – 3	0	act_0
R_1	0 – 1	14 – 15	1	2	0	act_0
R_2	0 – 1	8 – 11	0 – 3	2	1	act_1
R_3	0 – 1	8 – 11	0 – 3	1	1	act_2
R_4	0 – 1	8 – 11	2	3	1	act_1
R_5	0 – 7	14 – 15	2	1	0	act_0
R_6	0 – 7	14 – 15	2	2	0	act_0
R_7	0 – 7	8 – 15	0 – 3	0 – 3	1	act_2
R_8	0 – 15	4 – 7	0 – 3	0 – 3	1	act_2
R_9	0 – 15	0 – 7	0 – 3	1	0	act_0
R_{10}	0 – 15	0 – 15	0 – 3	0 – 3	0	act_3
R_{11}	0 – 15	0 – 15	0 – 3	0 – 3	1	act_4

Figure 4: A range based representation of the example with 12 rules on five fields shown in Figure 2.

Bucket size = 4

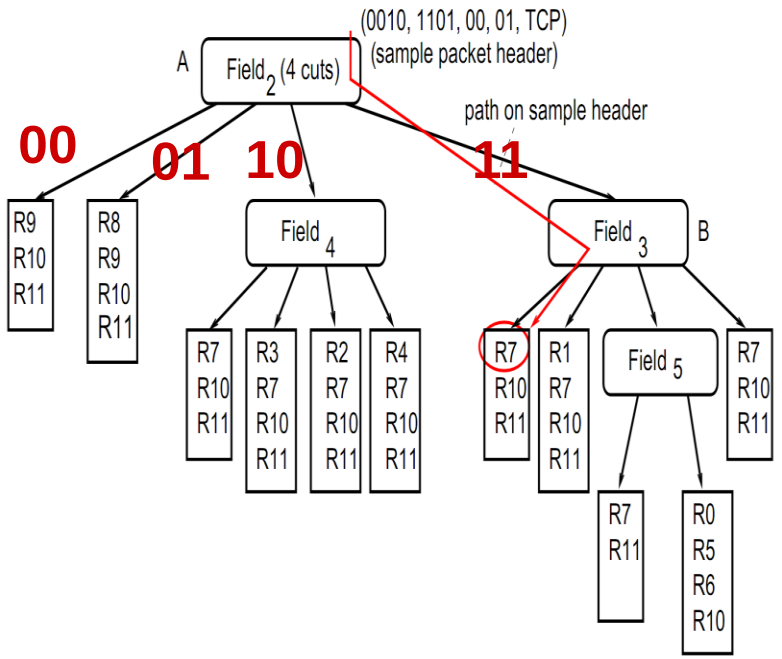
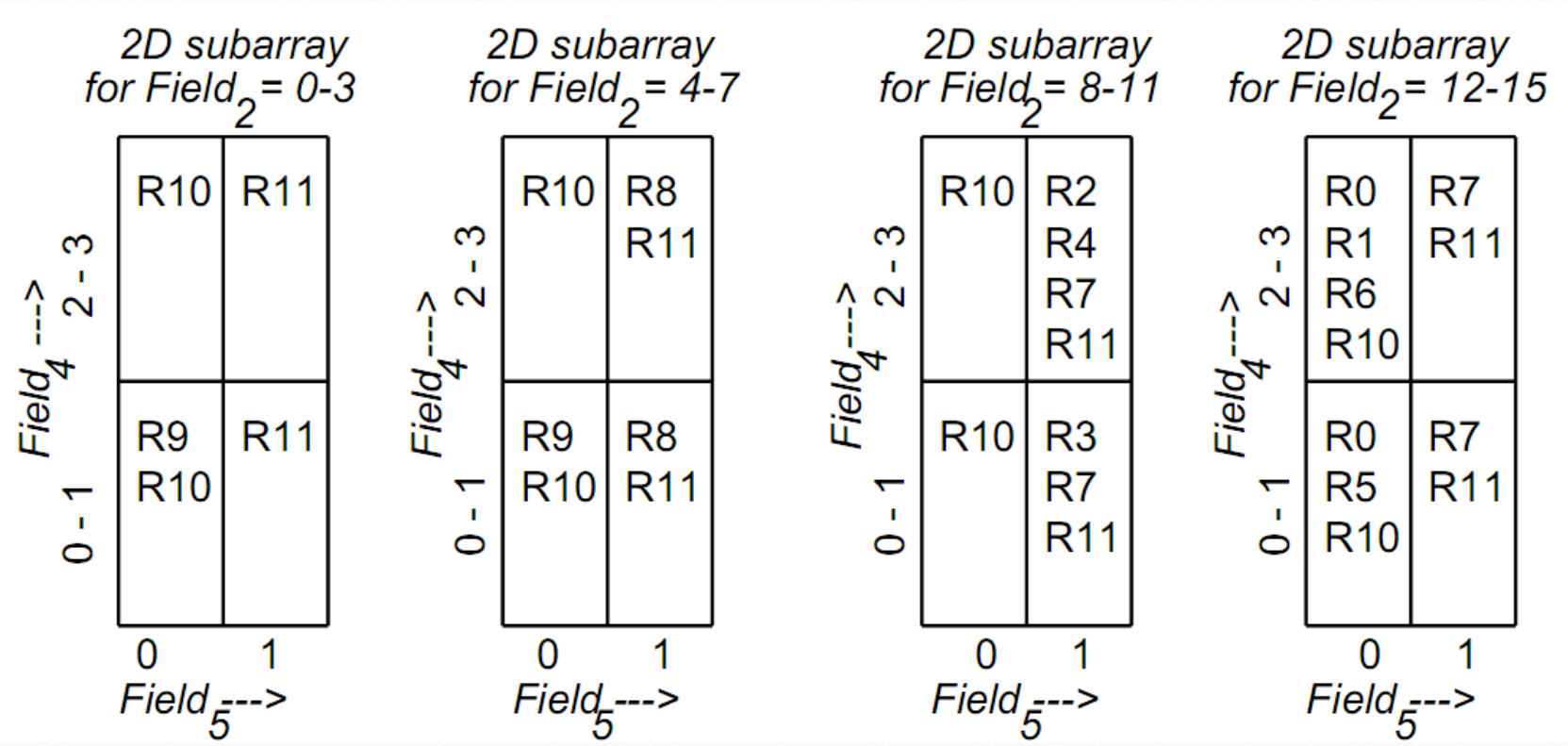


Figure 3: A HiCuts decision tree built for the database of Figure 2. $Field_2$ is chosen for the root node because it has the largest number of unique values. A sample packet header and a sample search path are also shown.

HyperCuts Example

Bucket size = 4

- The HyperCuts decision tree built using 4 cuts on *Field*2, and 2 cuts each on *Field*4 and *Field*5.
- Contrast this single node tree with Figure 5 which shows that any HiCuts tree must have height at least two.



HiSplit and HyperSplit

1. Similar to HiCuts and HyperCuts
2. Endpoints of ranges instead of prefixes
3. Cut the decision tree at endpoints instead of bits

Openflow Protocol

- **Software Defined Network (SDN)**

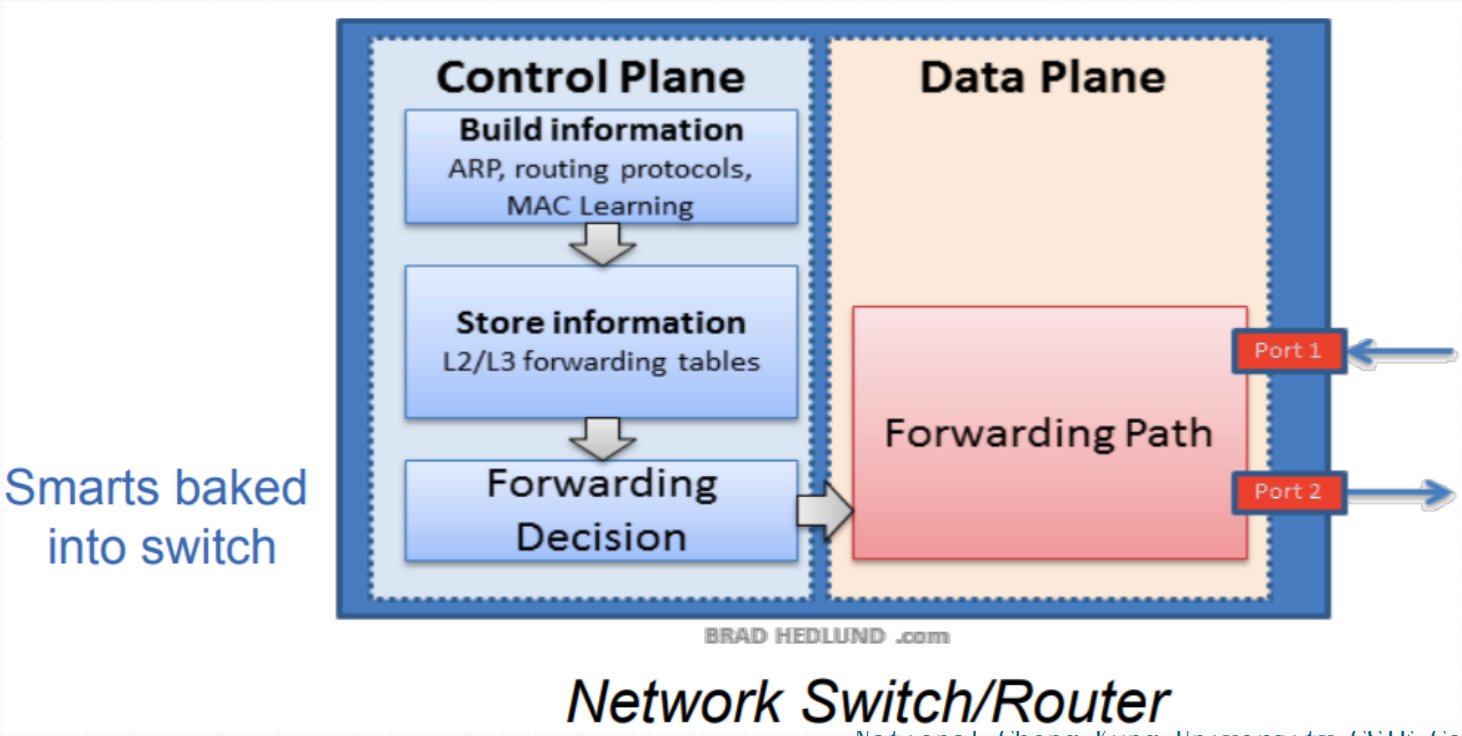
Field No.	1	2	3	4	5	6
Content	Source IP address	Destination IP address	Source transport port	Destination transport port	IP protocol	Ingress port
Size(bit)	32	32	16	16	8	6
Type	Prefix	Prefix	Range	Range	Exact	Exact

Field No.	7	8	9	10	11	12
Content	Source MAC address	Destination MAC address	Ethernet type	VLAN ID	VLAN priority	IP Tos
Size(bit)	48	48	16	12	3	6
Type	Exact	Exact	Exact	Exact	Exact	Exact

Total : 243 bits / packet header

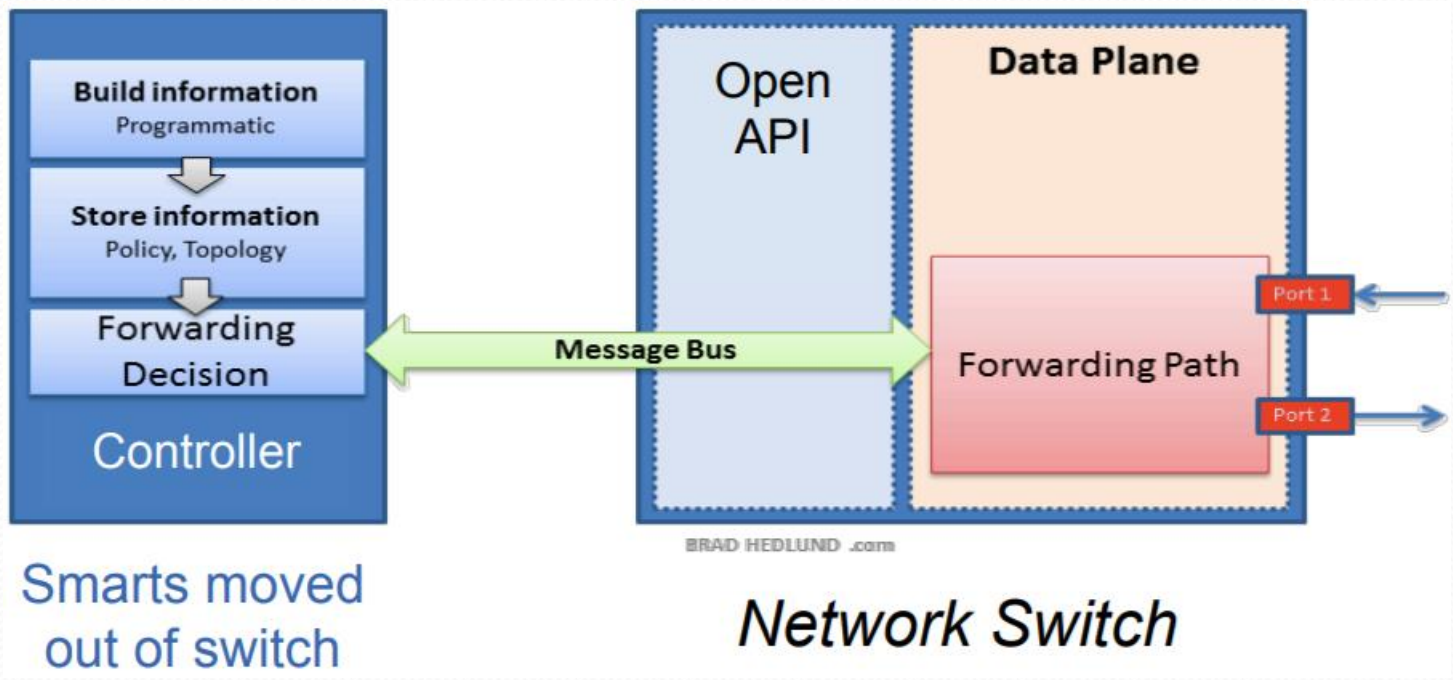
Drawbacks of Traditional Network

- Difficult to perform real world experiments on large scale production networks.
- Usage of custom ASICs with vendor specific software leads to innovation and configuration problems.



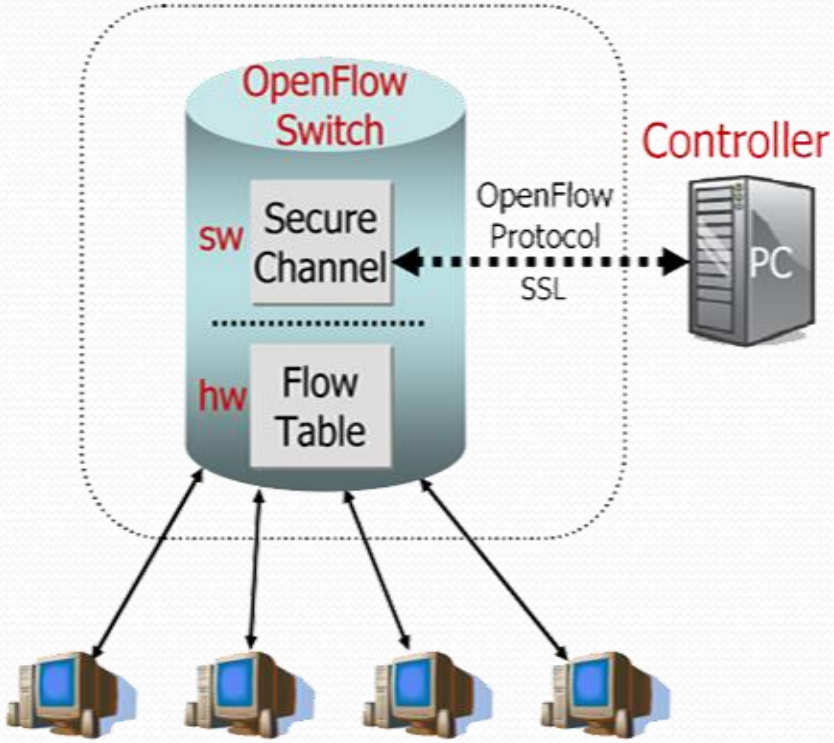
Software Defined Network

- Decouples the control plane and data plane.
- Program a network instead of configure a network.



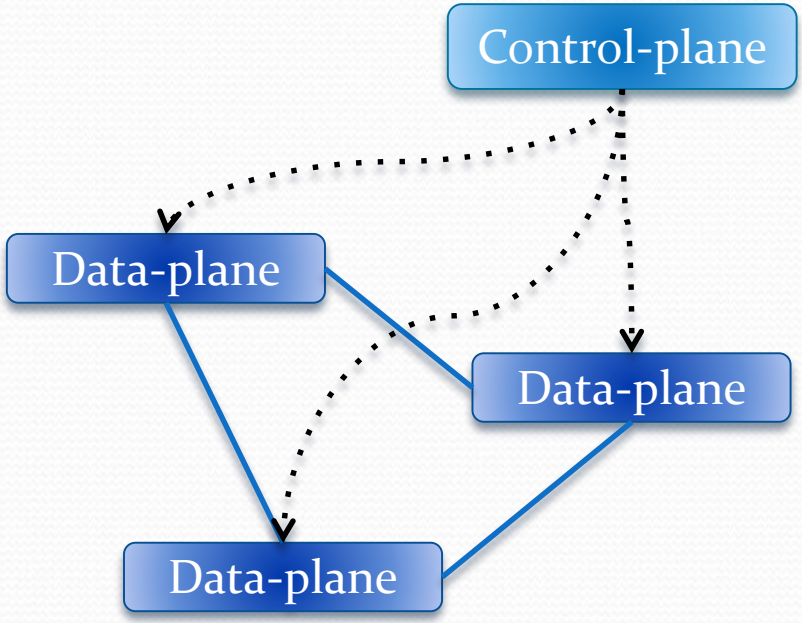
OpenFlow Overview

- Stanford Nick McKeown proposed a way for researchers to run experimental protocols in networks they use every day
- OpenFlow is a protocol which enables programmability of the data/forwarding plane.
- OpenFlow specification describes the requirements of an OpenFlow Logical Switch.
- OpenFlow specification also specifies a list of OpenFlow messages/API.



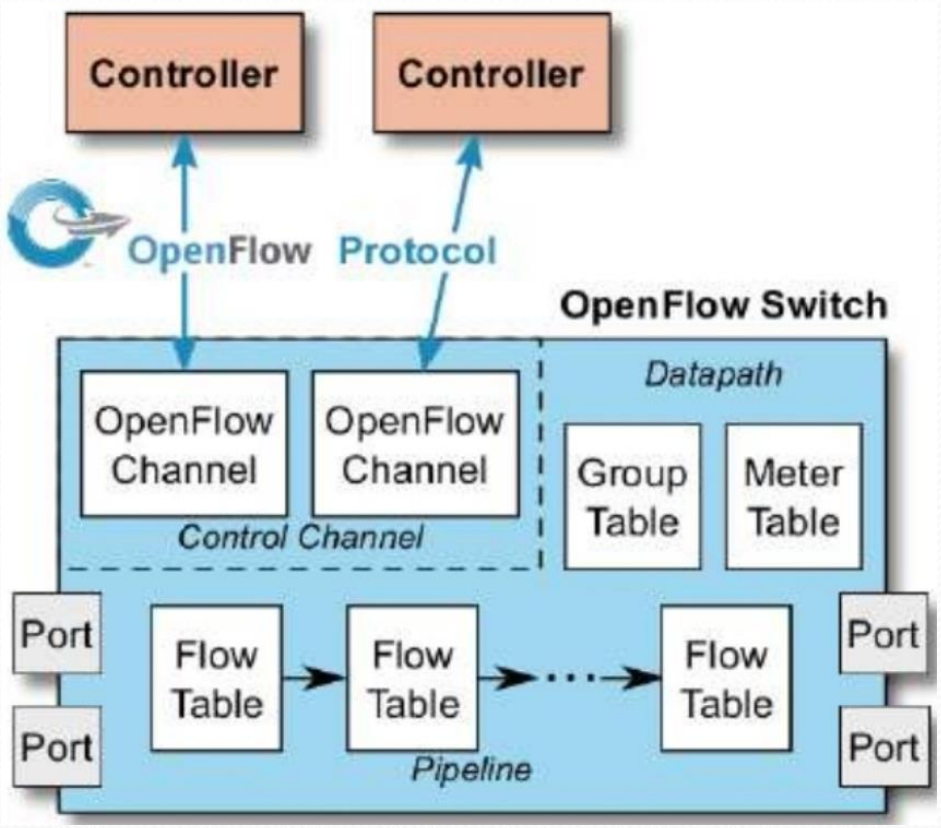
OpenFlow Controller

- Manages one or more switches via OpenFlow channels.
- OpenFlow protocol to communicate with OpenFlow switch.
- Provides a network wide abstraction for the applications on north bound.
- Responsible for programming various tables in OpenFlow switch.
- OpenSource controllers:
  - ONOS (JAVA)
  - OpenDayLight (JAVA)
  - Floodlight (JAVA)
  - RYU (Python)
  - NOX/POX (Python)

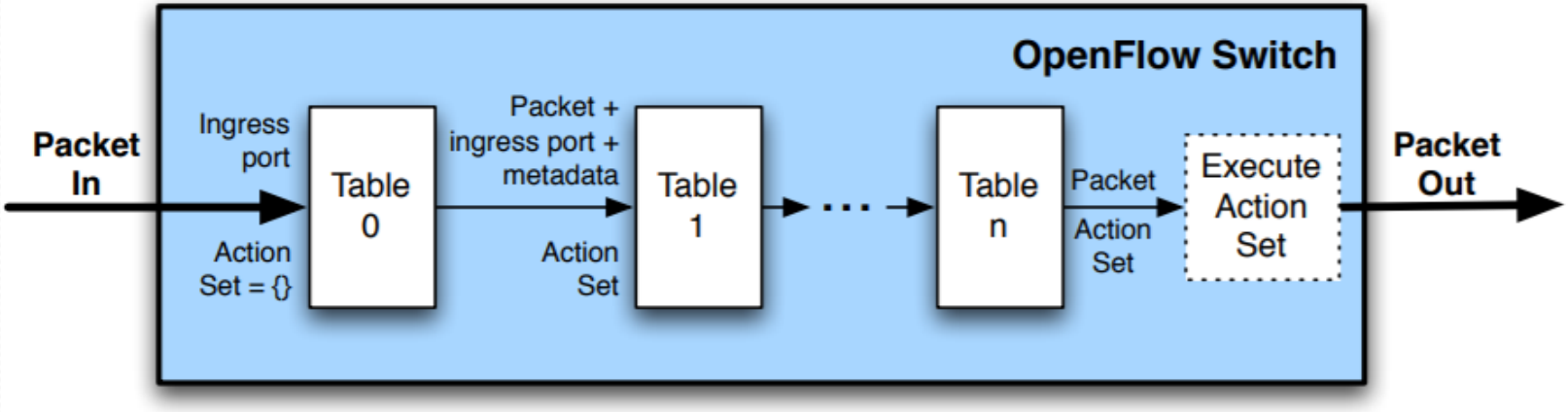


OpenFlow Aware Switch

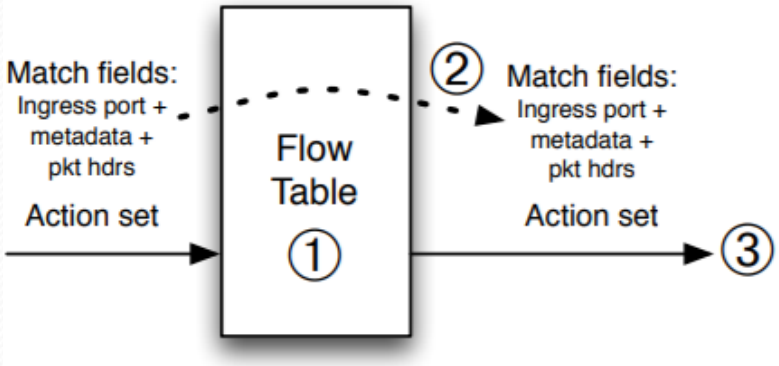
- OpenFlow 1.1 introduces multi table and group table support.
- OpenFlow 1.3 introduces meter table support.
- Up to 1.5.1



Packet Processing Pipeline



(a) Packets are matched against multiple tables in the pipeline



① Find highest-priority matching flow entry

② Apply instructions:
- i. Modify packet & update match fields (apply actions instruction)
- ii. Update action set (clear actions and/or write actions instructions)
- iii. Update metadata

③ Send match data and action set to next table

Flow Table

- A flow table consists of flow entries:

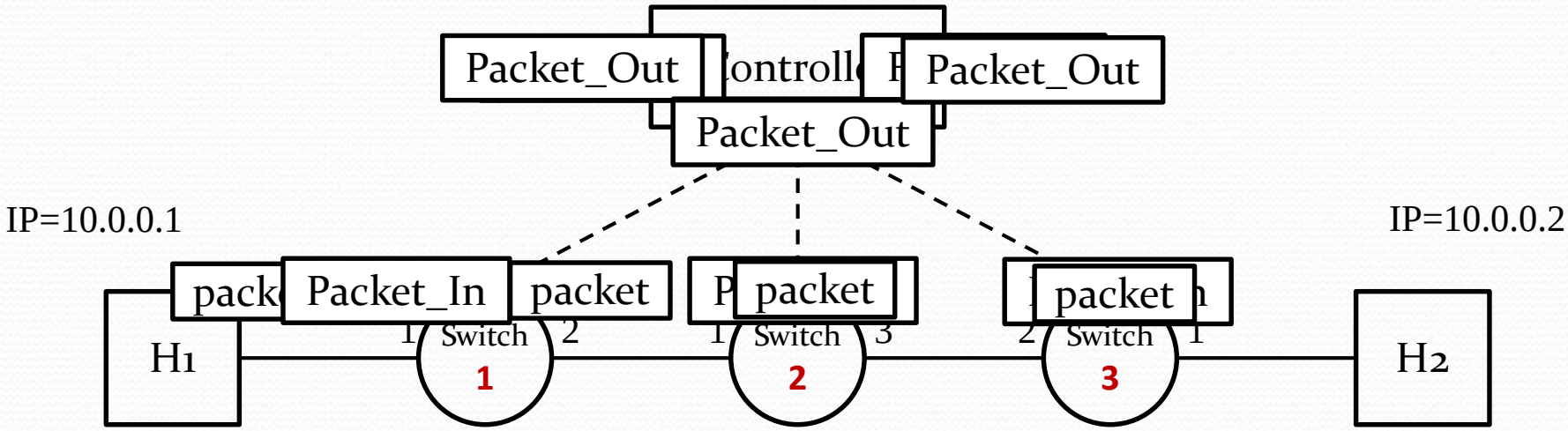
Match Fields	Priority	Counters	Instructions	Timeouts

- Match fields specifies which packet headers are used to match against. OpenFlow 1.0 supports 12 match fields, while OpenFlow 1.3 supports up to 40 match fields.

Switch Port	VLAN ID	VLAN prio	MAC src	MAC dst	Eth type	IP Src	IP Dst	IP ToS	IP Prot	L4 sport	L4 dport
16 in ovs	12	3	48	48	16			6			

- Priority describes the rule precedence.
- Counters field is used to count packets/bytes that match the entry.
- Instructions are executed when a packet matches the entry. Instructions contain either a set of actions to add to the action set, contains a list of actions to apply immediately to the packet, or modifies pipeline processing.
- Timeouts specify the liveness time of a entry.

Reactive Forwarding Scheme



Src_IP	Dst_IP	Out
10.0.0.1	10.0.0.2	2
10.0.0.1	10.0.0.2	123

Src_IP	Dst_IP	Output
10.0.0.1	10.0.0.2	1
10.0.0.1	10.0.0.2	123

OpenFlow Secure Channel

- Used to exchange OpenFlow message between switch and controller.
- Switch can establish single or multiple connections to different controllers.
- A controller configures and manages the switch, receives events from the switch, and send packets out the switch via this interface.
- The SC connection is a TLS/TCP connection.

OpenFlow Messages

- Controller-to-Switch - initiated by the controller and used to directly manage or inspect the state of the switch.

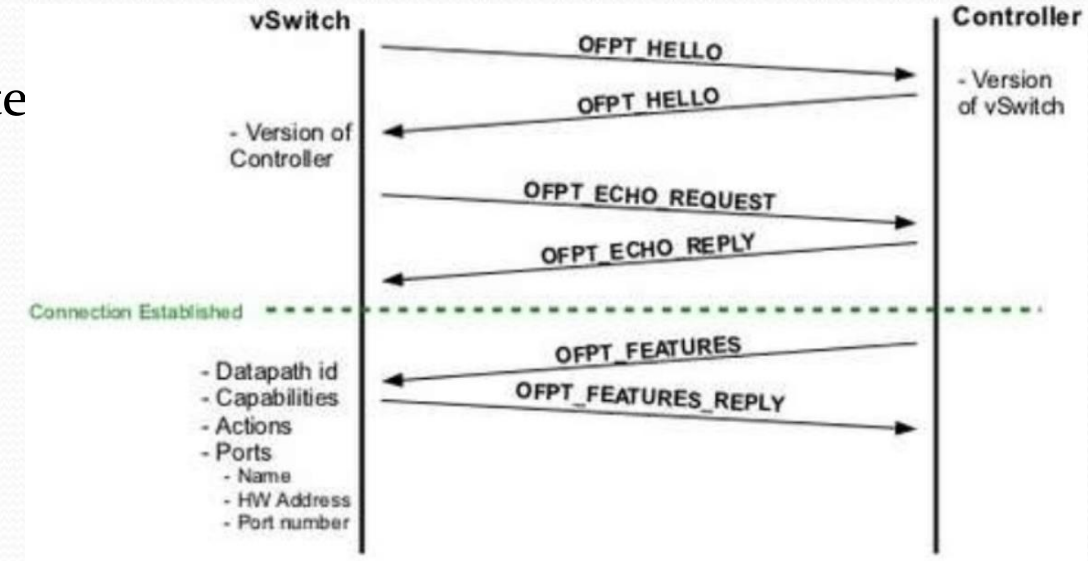
  Features, Configure, Modify State, Read-State, Packet-Out, Barrier

- Asynchronous - Asynchronous messages are sent without the controller soliciting them from a switch.

  Packet-in, Flow Removed/Expiration, Port-Status

- Symmetric – Symmetric messages are sent without solicitation, in either direction.

  Hello, Echo, Experimente



OpenFlow Instructions and Actions

- Instruction types:

  Apply-Actions – Apply a specific action list immediately.

  Clear-Actions – Clear all actions in the action set.

  Write-Actions – Add a new action into the existing action set.

  Write-Metadata – Write the masked meta data value.

  Goto-Table – Indicate the next table in the processing pipeline.

  Meter ID – Direct a packet to the meter ID.

- Action types:

  Output – Forward a packet to the specified port.

  Drop – Drop a packet.

  Set-Field – Rewrite a field in the packet header.

  Change TTL – Decrement TTL.

  Push MPLS – Apply MPLS tag push action to the packet.

Programming Protocol-Independent Packet Processors(P4)

- Why P4?

  OpenFlow match fields keep growing and growing.

  OpenFlow multi table support leads to vendor lock-in.

- What is P4?

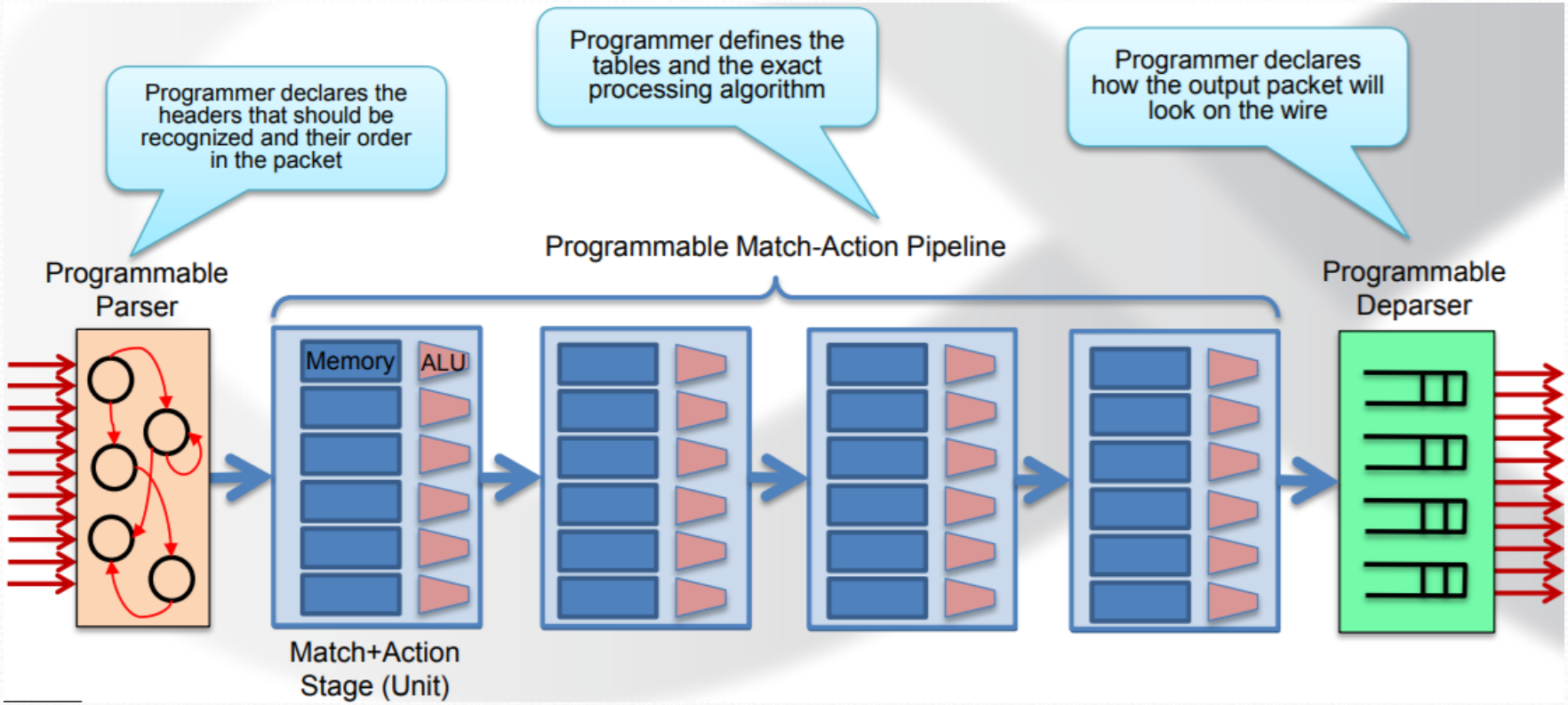
  P4 is a language designed to allow top to bottom programmability. P4 is very similar to C language.

- How does P4 work?

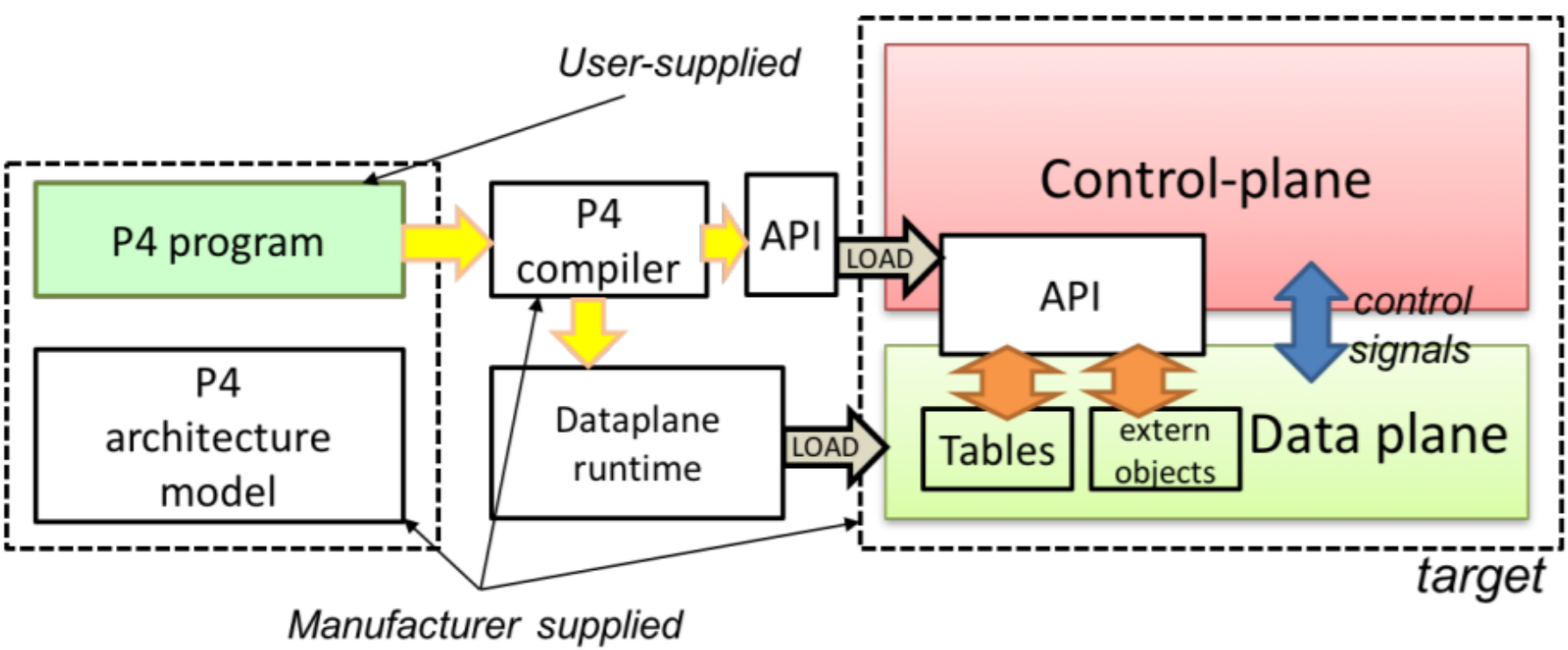
  Programmable network devices like Barefoot Tofino and Xilinx NetFPGA.

P4 Features and PISA

- Protocol Independence Switch Architecture
- Target Independence
- In-field re-configurability

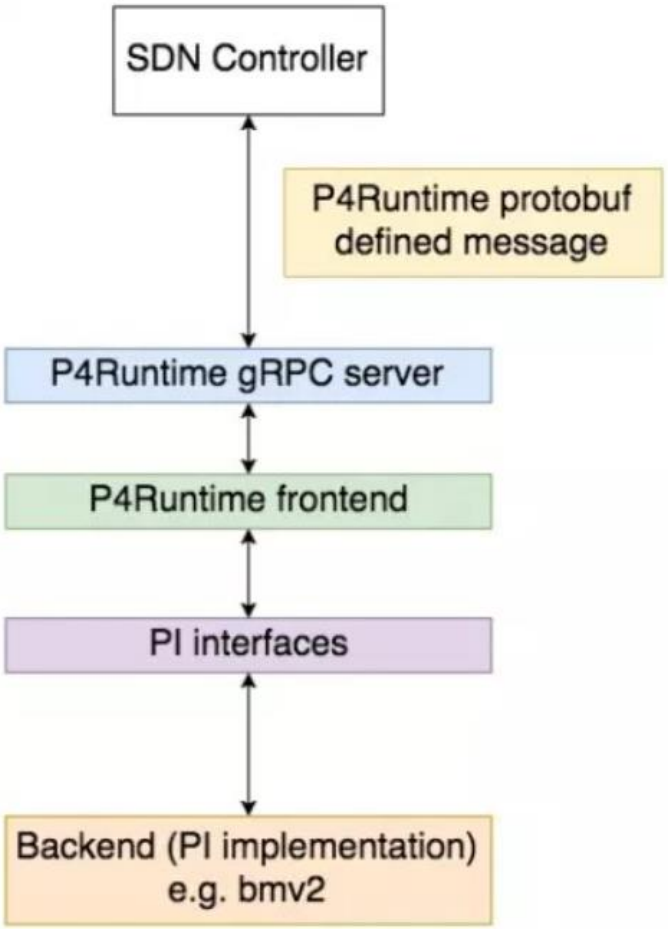


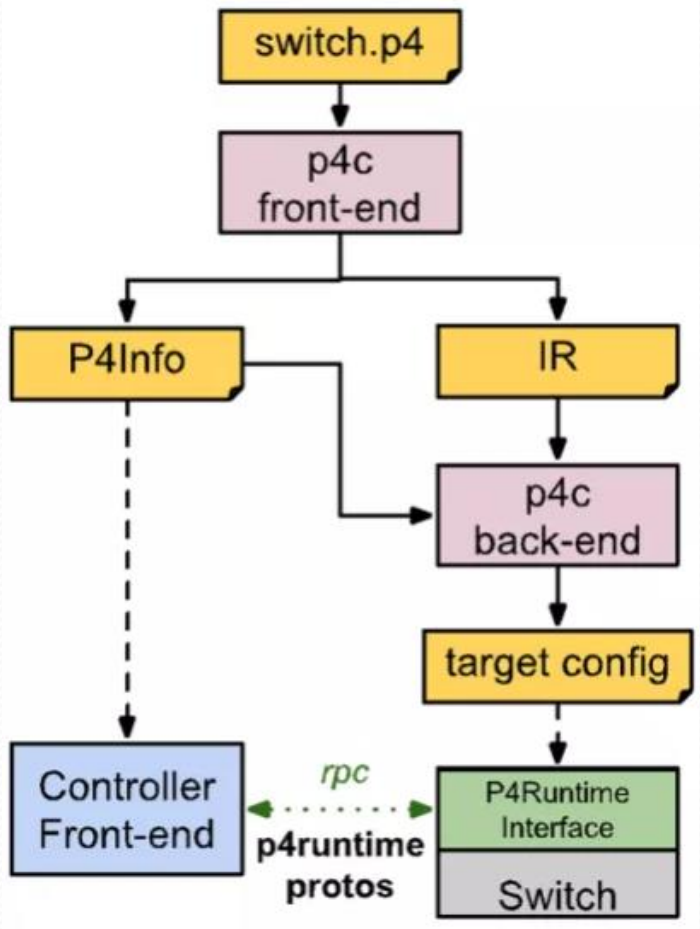
P4 Workflow


User-supplied
P4 program
P4 architecture model
P4 compiler
API
Dataplane runtime
LOAD
LOAD
Control-plane
API
control signals
Tables
extern objects
Data plane
target
Manufacturer supplied

P4 Runtime

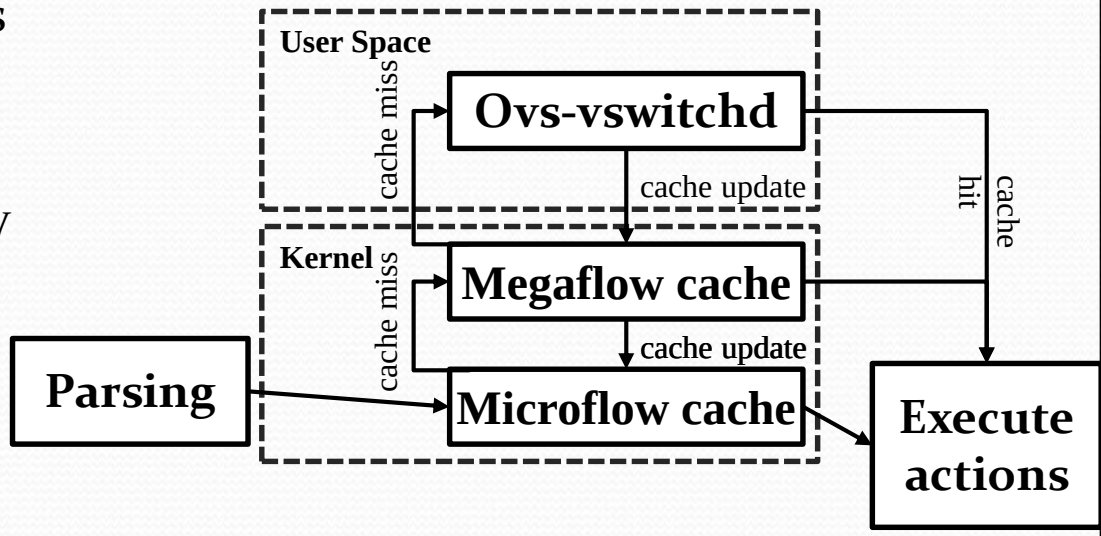
- P4 runtime is a new way (protocol) for control plane software to communicate with data plane.
- P4 runtime is based on protocol buffer and grpc.





Open vSwitch

- OVS is an open-source, software implementation of a virtual multilayer network switch, designed for network management and protocol supports, such as NetFlow and sFlow.
- A variant of TSS called Priority Sorting Tuple Space Search (PSTSS) is implemented on megaflow cache.



PSTSS & TupleMerge

- TSS partitions rules into a set of tuples, each tuple can be regard as a concatenation of the actual bits used in each field, so when a packet comes, the bits extracted from the parser will form a hash key to those tuples for searching.
- PSTSS improves the exhaustive search of all tuples by recording the highest priority rule for each tuple, so the process can be terminated if the priority of matching rule is higher than the one stored in next tuple.
- TupleMerge merge the tuples by relaxing the restrictions. Rules in these tuples have similar characteristics are combined into a bigger one.

A 2-field classifier			
Rule	Priority	Field X	Field Y
R1	7	000*	001*
R2	6	010*	10**
R3	5	101*	00**
R4	4	11**	01**
R5	3	001*	01**
R6	2	10**	1***
R7	1	01**	0***

PSTSS		
Tuple	Tup. Pri.	Hash Table Entries
(3, 3)	7	{R1}
(3, 2)	6	{R2, R3, R5}
(2, 2)	4	{R4, R6}
(2, 1)	1	{R7}

Tuple Merge		
Tuple	Tup. Pri.	Hash Table Entries
(3, 2)	7	{R1, R2, R3, R5}
(2, 1)	4	{R4, R6, R7}

Proposed Scheme - Motivation

- Decision tree based: Search fast, but update slow.
- Tuple Space based: Search slow, but update fast.
- When the size of the ruleset becomes very large, the search performance of both are severely affected.
- Two ways to partition the classifier:
  - Using the field values of the rules (the first k bits of prefix field, protocol number)

    → The packet also has the information, when the field values are used to divide the subsets, the subsets are independent of each other.
  - Using the additional information of rules (prefix length, wildcard or non-wildcard)

    → All tuples need to search since the packet does not have these information.

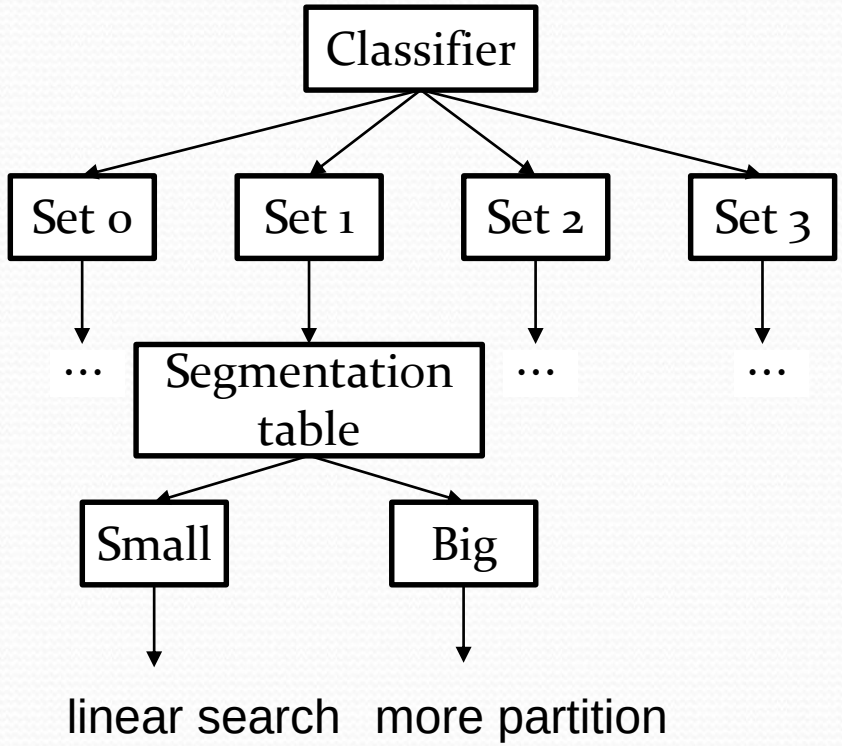
Proposed Scheme - Overview

- A two-stage architecture:

Partitioning
based on prefix length

Partitioning
based on field values

Using method
based on threshold

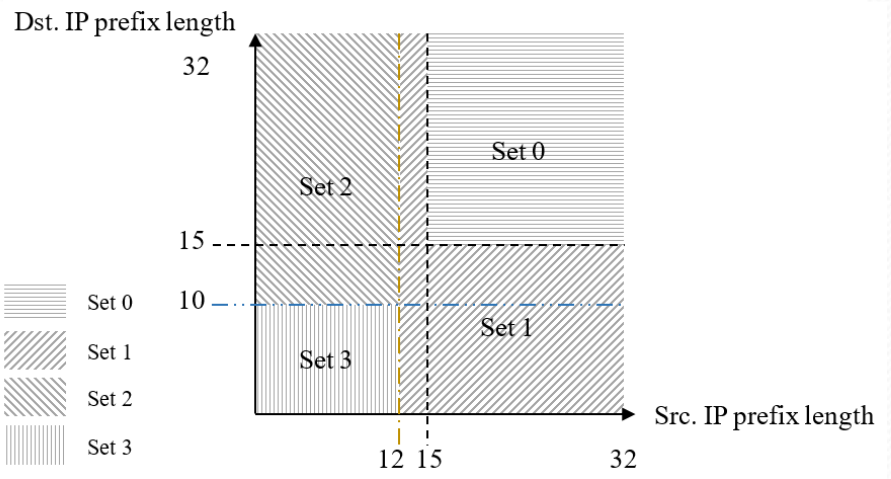


Proposed Scheme –Stage 1(1/3)

Basic rule partitioning:

- We need to determine the prefix length of each set to allocate the rules to the four sets.
- We also need to allocate an appropriate size for each segmentation table.
- $Used\ bits = log_2(\#\ of\ rules\ in\ set)$

- Partition based on new prefix length



Proposed Scheme –Stage 1(2/3)

- **Enhanced rule partitioning:**
  - **Set 0:** Prefix length of source IP address ≥ k_0 and Prefix length of destination IP address ≥ k_0.
  - **Set 1:** Prefix length of source IP address ≥ k_1.
  - **Set 2:** Prefix length of destination IP address ≥ k_2.
  - **Set 3:** None of the above.
- **Partition rules into segments:**
  - **Set 0:** (concatenate first k_0 $bits$ of src. IP and dst. IP $*$ magic value)
  - **Set 1:** (first k_1 $bits\ of\ src.IP$)
  - **Set 2:** (first k_2 $bits\ of\ dst.IP$)

Proposed Scheme –Stage 1(3/3)

Tuple	Rule list	Tuple	Rule list
(6, 6)	{R1, R2, R3}	(3, 2)	{R13, R14}
(5, 6)	{R4, R5}	(2, 3)	{R15 }
(5, 5)	{R6}	(1, 3)	{R16}
(5, 4)	{R7, R8}	(2, 0)	{R17}
(4, 4)	{R9}	(1, 1)	{R18}
(3, 3)	{R10}	(0, 1)	{R19}
(4, 2)	{R11, R12}	(1, 0)	{R20}

Step 1: The basic rule partition based on prefix length of 3.

$k_0 = \lfloor log_2 10 \rfloor = 3$,
$k_1 = \lfloor log_2 4 \rfloor = 2$
$k_2 = \lfloor log_2 2 \rfloor = 1$

	Rule list	# of rules
Set 0	{R1, R2, R3, R4, R5,R6, R7, R8, R9, R10}	10
Set 1	{R11, R12, R13, R14}	4
Set 2	{R15, R16}	2
Set 3	{R17, R18, R19, R20}	4

Step 2: Enhanced rule partition based on k_0=3, k_1=2, k_2=1.

	Rule list	# of rules
Set 0	{R1, R2, R3, R4, R5,R6, R7, R8, R9, R10}	10
Set 1	{R11, R12, R13, R14, R15, R17}	6
Set 2	{R16, R18, R19}	3
Set 3	{R20}	1

Step 3: Partition rules into segment based on their filed values.

Proposed Scheme –Stage 2(1/3)

- According to the number of rules in each segment, we define a threshold T1 to divide segment into two types:
  - Small segment: number of rules in the segment ≤ T1.
  - Big segment: number of rules in the segment > T1.

- **Small segment:** We can simply use the linear search.

- **Big segment:** We must further perform partition step to reduce the number of rules to be searched.

Proposed Scheme –Stage 2(2/3)

- For the big segment, we generate up to five subsets:
  - **Subset 0:** Prefix length of source IP address ≥ 30.
  - **Subset 1:** Prefix length of destination IP address ≥ 30.
  - **Subset 2:** Destination port is an exact value.
  - **Subset 3:** Source port is an exact value.
  - **Subset 4:** None of the above.

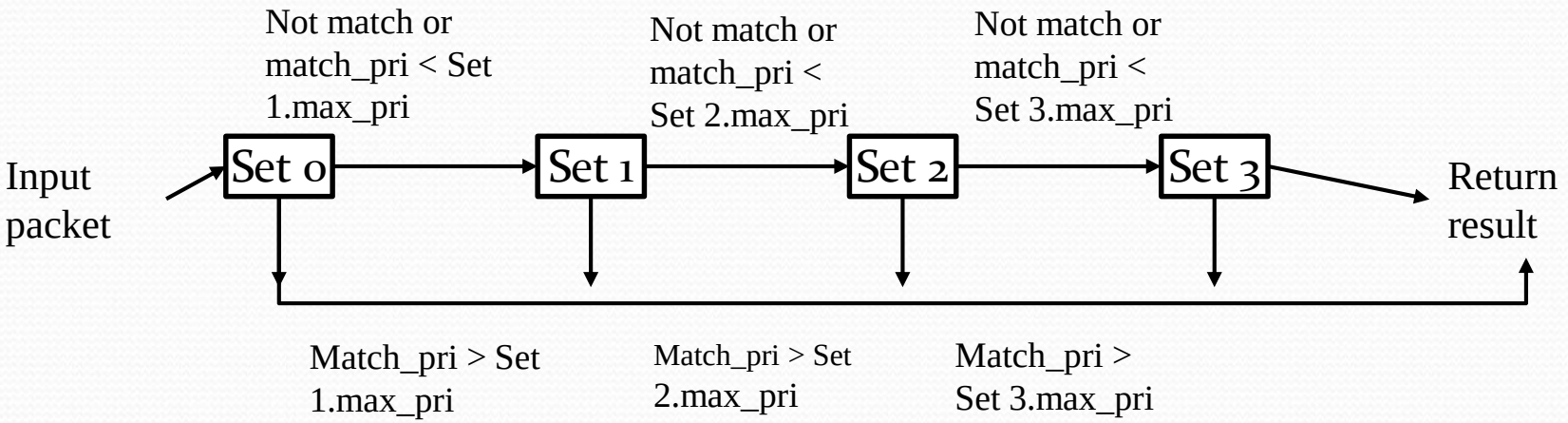
- We define a new threshold T2 to determine whether each subset belongs to a small subset or a big subset.

Proposed Scheme –Stage 2(3/3)

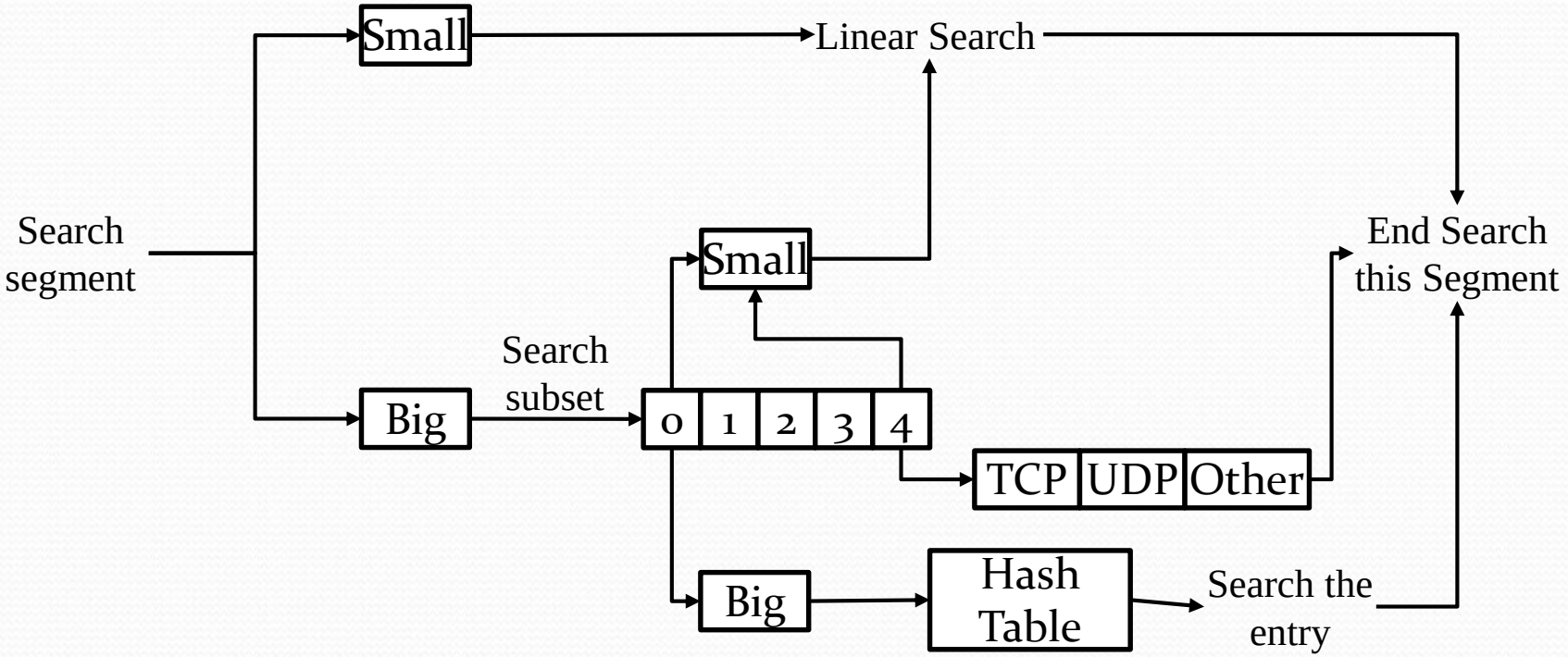
- **Small subset in big segment:** Linear search data structure is used in the same way as the small segment.
- **Big subset in big segment:** Create a hash table of appropriate size.
  - **Subset 0 (Big):** Use the first 30 bits of src. IP as the input of the hash function.
  - **Subset 1 (Big):** Use the first 30 bits of dst. IP as the input of the hash function.
  - **Subset 2 (Big):** Use the dst. port as the input of the hash function.
  - **Subset 3 (Big):** Use the src. port as the input of the hash function.
  - **Subset 4 (Big):** According to the protocol field, it is divided into tcp, udp, others, a total of 3 groups.

Proposed Scheme – Priority Sorting

- We pre-record the maximum priority among all the rules of each set.
- If the currently matched rule has a higher priority than the maximum priority of the next set, we can skip the search process in the next set.



Proposed Scheme – Searching the Segment



Experimental Results – Environment & Rulesets

- The specifications of the experimental environment.

Item	Description
CPU	Intel Core i5-4460 3.2GHz
RAM	DDR3 16GB
OS	Ubuntu 18.04 64-bit
C++ compiler	g++ 7.5.0

- The 12 rulesets that are generated by ClassBench with 12 seed files (i.e. 5 ACLs, 5 FWs, and 2 IPCs) contain about 100K rules and the sizes of their corresponding trace are 10 times the size of the classifiers, and other 7 fields in the exact format are added for multi-field classifiers based on OpenFlow 1.0.

Experimental Results – Analysis (1/3)

- Number of rules for each set before and after determining segment table size.

Befor

Rule Sets	# of rules	# of rules in each set			
		Set 0	Set 1	Set 2	Set 3
ACL1	99,833	98,044	1,568	0	221
ACL2	89,229	61,768	13,149	13,174	1,138
ACL3	99,859	84,306	5,192	9,912	449
ACL4	99,882	84,753	5,639	8,351	1,139
ACL5	99,409	98,137	0	1272	0
FW1	96,376	18,616	21,255	55,302	1,203
FW2	98,555	33,473	51,729	11,387	1,966
FW3	92,932	8,241	23,337	59,666	1,688
FW4	92,765	24,986	17,138	48,631	2,010
FW5	93,324	5,512	32,689	53,214	1,909
IPC1	99,782	88,457	4,807	6,205	313
IPC2	100,000	36,559	10,960	52,481	0

After

Rule Sets	# of rules	# of rules in each set			
		Set 0	Set 1	Set 2	Set 3
ACL1	99,833	98,044	1,570	72	147
ACL2	89,229	66,377	14,771	7,339	742
ACL3	99,859	84,306	8,974	6,386	193
ACL4	99,882	84,753	9,817	5,029	283
ACL5	99,409	98,137	1,163	109	0
FW1	96,376	18,616	21,392	55,302	1,066
FW2	98,555	33,915	53,399	10,945	296
FW3	92,932	8,241	23,502	59,666	1,523
FW4	92,765	24,986	17,138	48,631	2,010
FW5	93,324	5,512	32,927	53,214	1,671
IPC1	99,782	88,457	5,649	5,572	104
IPC2	100,000	36,559	10,960	52,481	0

Experimental Results – Analysis (2/3)

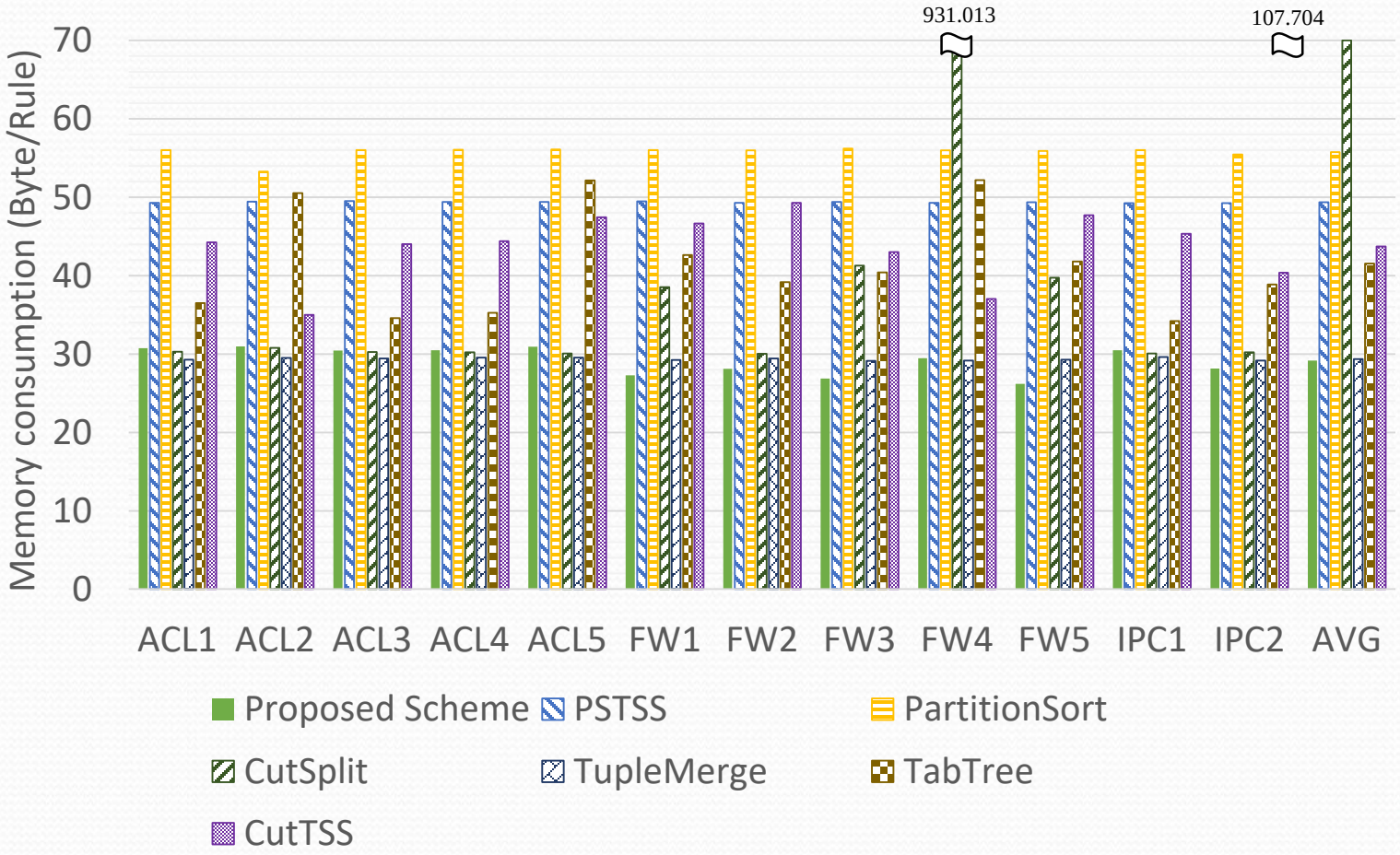
- Priority Sorting

Rule Sets	The percentages of rules in the first i sets, for i=0 to 2 that have a higher priority than the maximum priority of set i+1		
	Set 0	Set 0 + Set 1	Set 0 + Set 1 + Set 2
ACL1	99.047	99.989	99.963
ACL2	31.125	26.469	91.269
ACL3	83.855	83.153	98.548
ACL4	83.242	82.991	95.215
ACL5	94.516	99.483	100.000
FW1	25.967	14.667	99.502
FW2	83.633	59.780	57.083
FW3	100.000	98.821	18.098
FW4	84.415	86.139	32.878
FW5	81.948	11.923	98.885
IPC1	57.796	59.519	97.584
IPC2	100.000	100.000	44.838

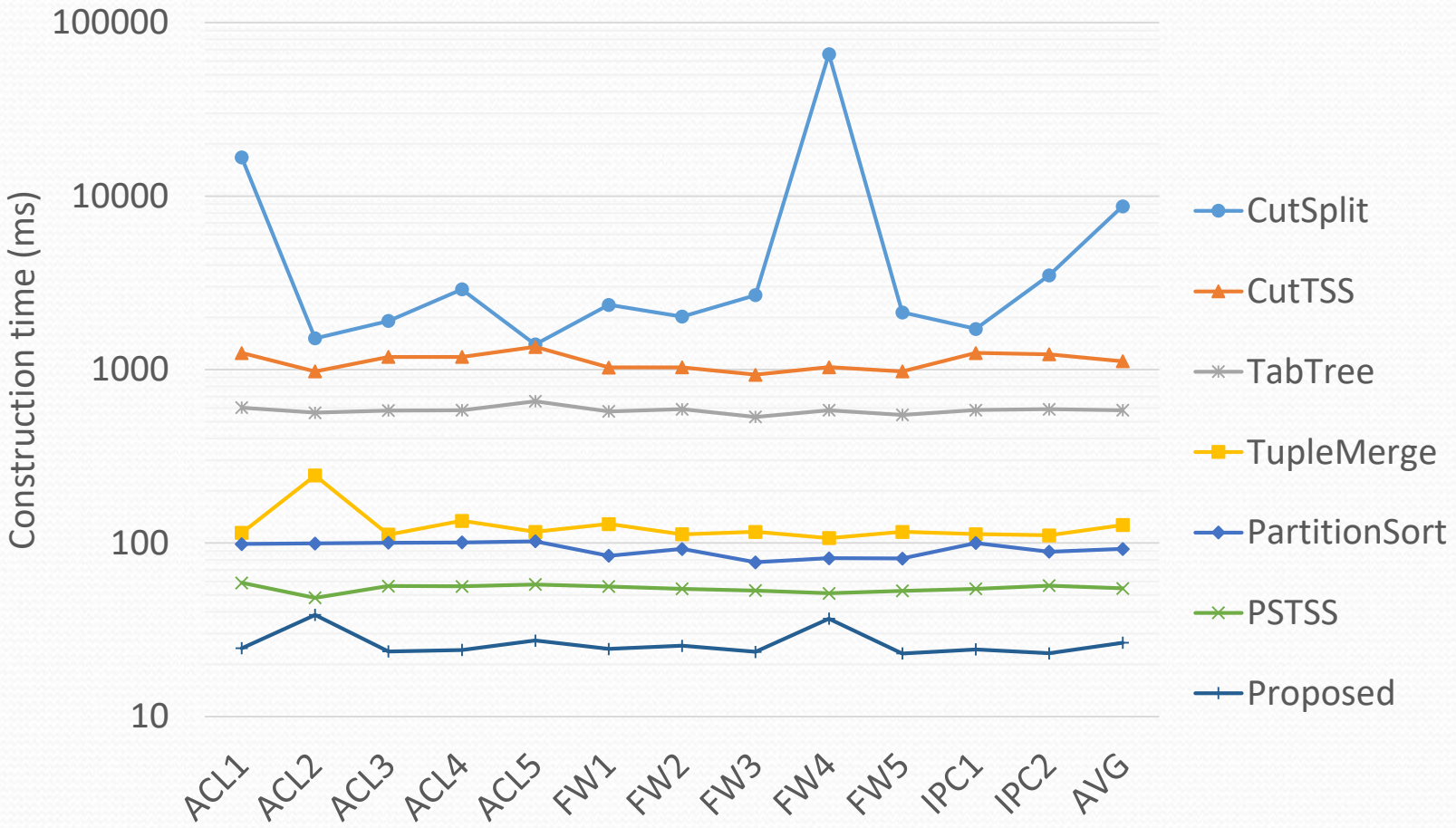
Experimental Results – Analysis (3/3)

Rule Sets	# of small and big segments						Total	
	Set 0		Set 1		Set 2			
	Small	Big	Small	Big	Small	Big	Small	Big
ACL1	46,902	283	680	2	6	2	47,588	287
ACL2	5407	1,204	467	343	223	135	6,097	1,682
ACL3	42,778	5	3,677	0	2,284	295	48,739	300
ACL4	45,243	2	3,758	0	642	175	49,643	177
ACL5	28,518	745	251	3	89	0	28,858	748
FW1	4,214	263	1,732	375	26,287	0	32,233	638
FW2	6,877	582	15,291	83	291	931	22,459	1,596
FW3	323	245	636	338	16,024	0	16,983	583
FW4	120	664	231	552	3,989	1,275	4,340	2,491
FW5	1,271	97	4,121	545	29,612	0	35,004	642
IPC1	43,299	14	3,069	0	1,483	151	47,851	165
IPC2	5,282	884	3,898	0	8,176	16	17,356	900

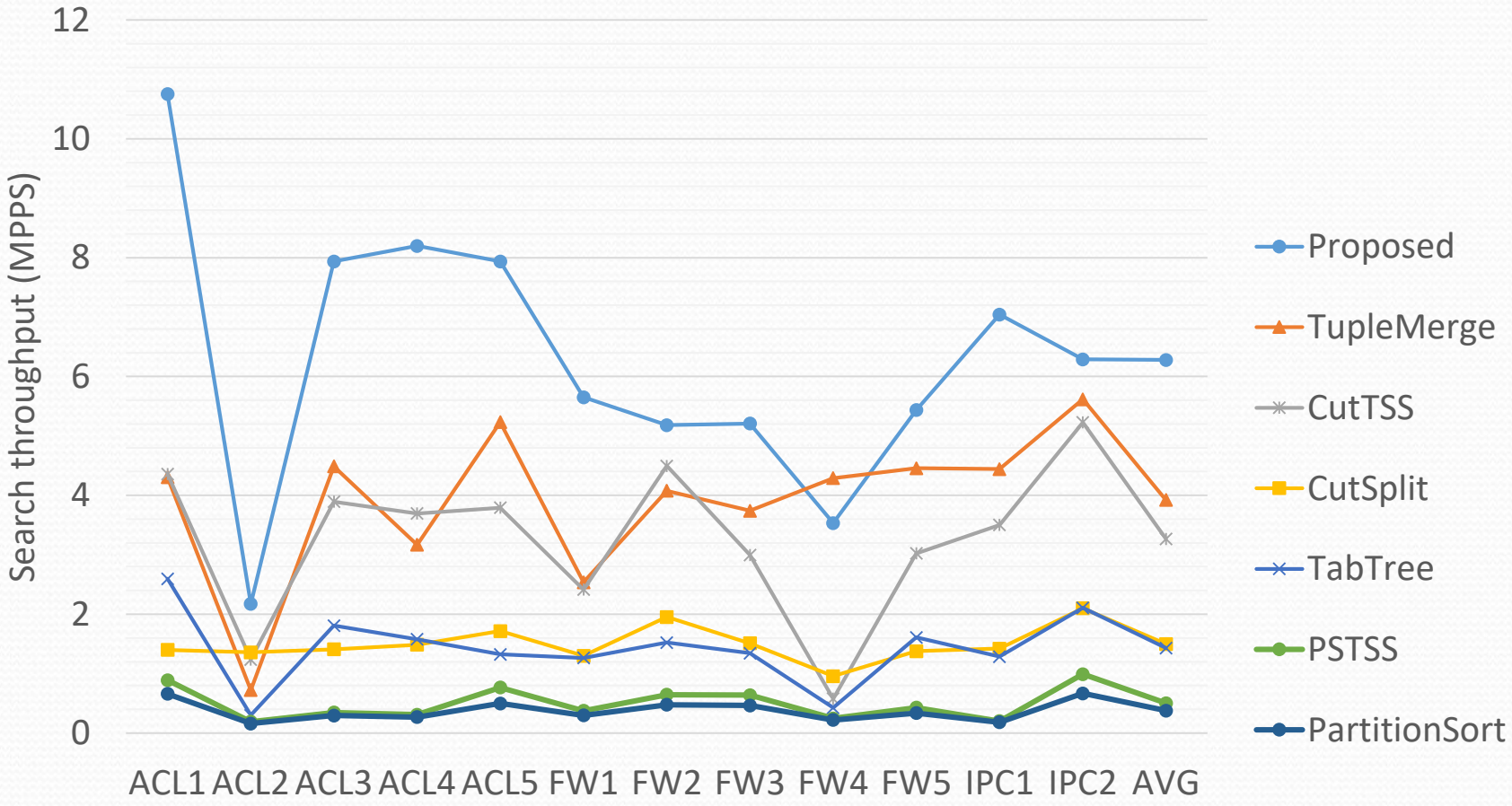
Experimental Results – Memory Consumption



Experimental Results – Construction Time



Experimental Results – Classification Performance (1/2)

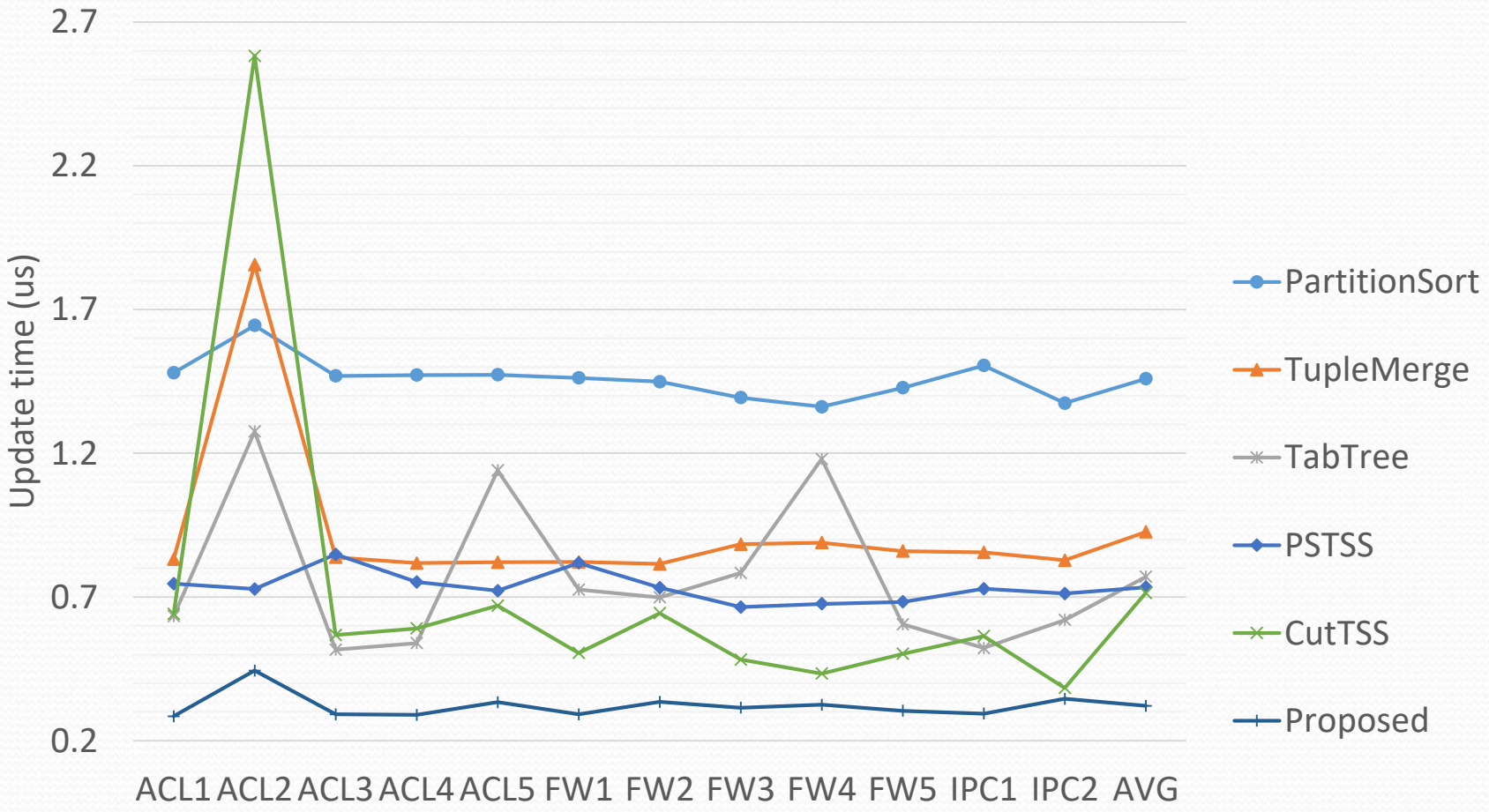


Experimental Results – Classification Performance (2/2)

- Performance impact $= \frac{10\%\ search\ missing - 100\%\ search\ hit}{10\%\ search\ missing} \times 100\%$

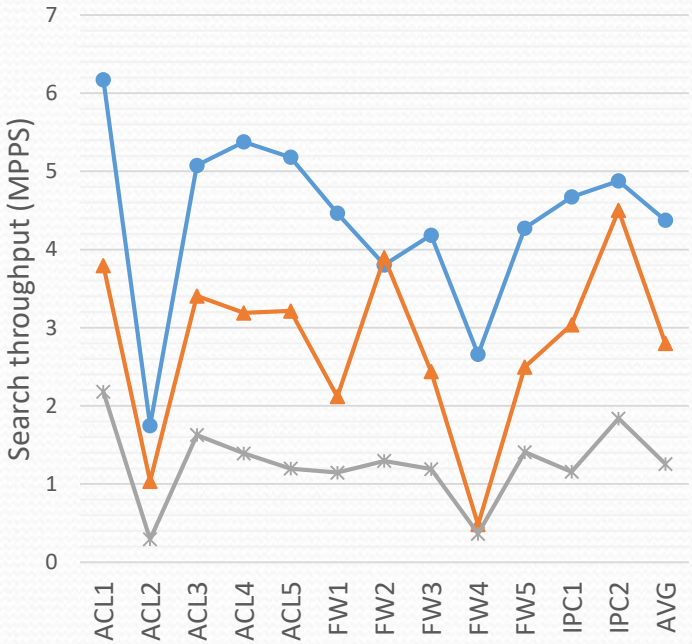
Scheme	Classification throughput (MPPS)		Performance impact (%)
	100% search hit	10% search missing	
Proposed	6.278	5.783	-7.889
PSTSS	0.502	0.425	-15.418
PartitionSort	0.377	0.327	-13.221
CutSplit	1.499	1.514	1.014
TupleMerge	3.923	3.695	-5.811
TabTree	1.431	1.269	-11.327
CutTSS	3.268	2.730	-16.462

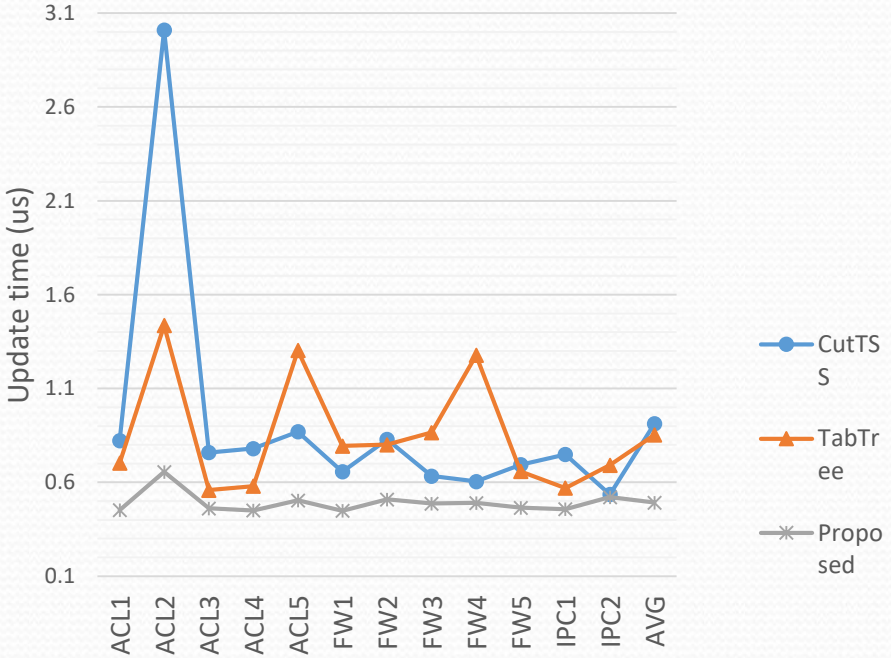
Experimental Results – Update time



Experimental Results - Multi-field

- Classification performance
- Update time





Thanks for Your Attention