

High-Speed Router SoC

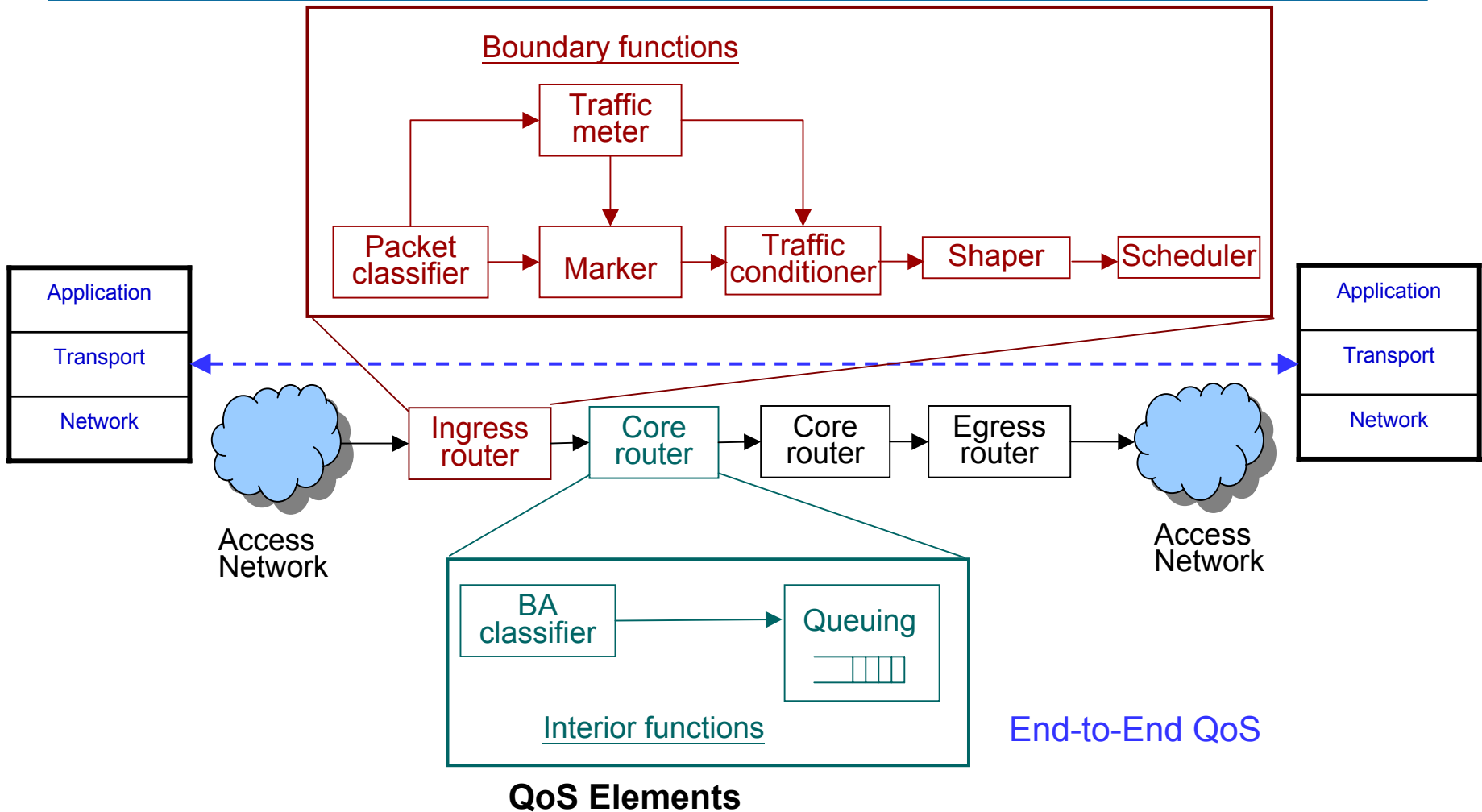
李程輝

OUTLINE

- QoS Router Architecture
- IP Address Lookup Algorithms
- Leaky Bucket Algorithm
- Weighted Fair Queuing
- Summary

QoS Router Architecture

QoS Elements



Router Basic Functions

- **Packet Forwarding**

 - IP Route Table Lookup

 - ARP

- **Address Translation**

 - NAT/PAT

QoS Elements

- **Packet Classifier**

 - IPv4 Multi-Field-5

- **Meter/Marker**

 - Token Bucket Meter (srTCM, trTCM)

- **Dropper**

 - Weighted Random Early Detection (WRED)

- **Shaper**

 - Token Bucket Meter

- **Scheduler**

 - Strict Priority+Weighted Fair Queuing (SP/WFQ)

Network Security

■ IPsec

AH (MD5 & SHA_1) + ESP (DES, 3DES, AES)

PRNG + RSA + ECC

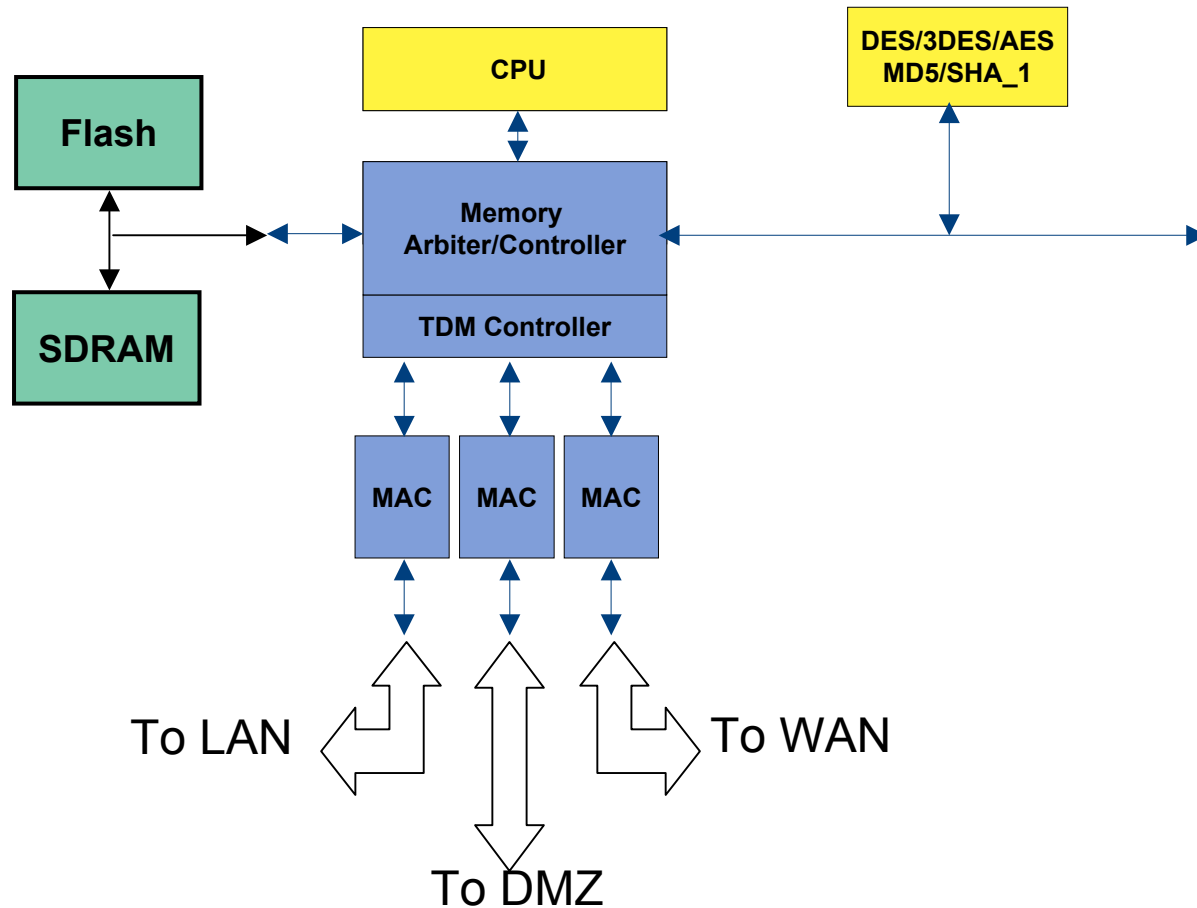
■ Firewall

Packet Filtering

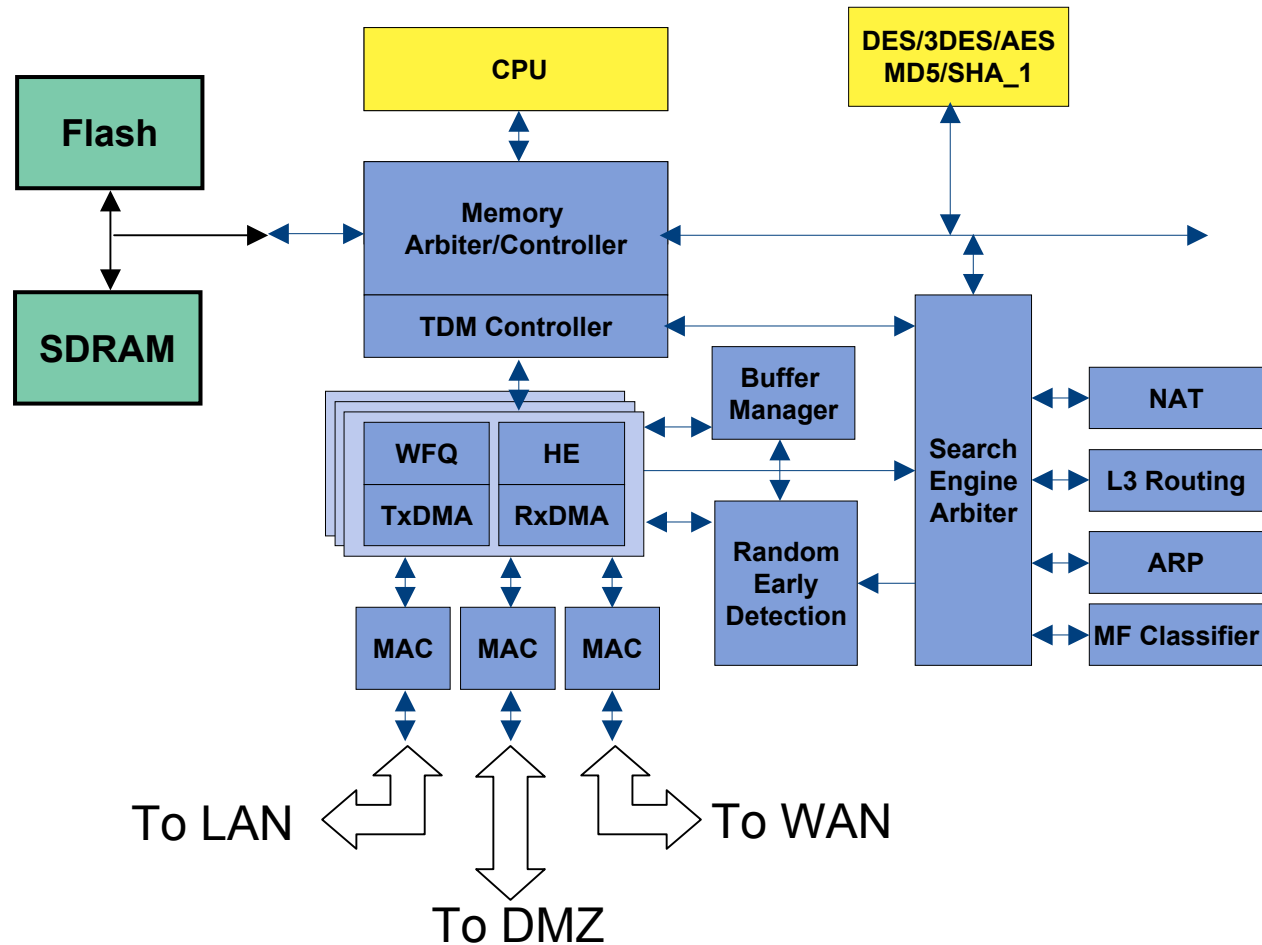
Proxy

Stateful Packet Inspection

Software Based SoC Architecture



High-Speed Router SoC Architecture



IP Address Lookup Algorithms

Example Forwarding Table (5-bit Prefixes)

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

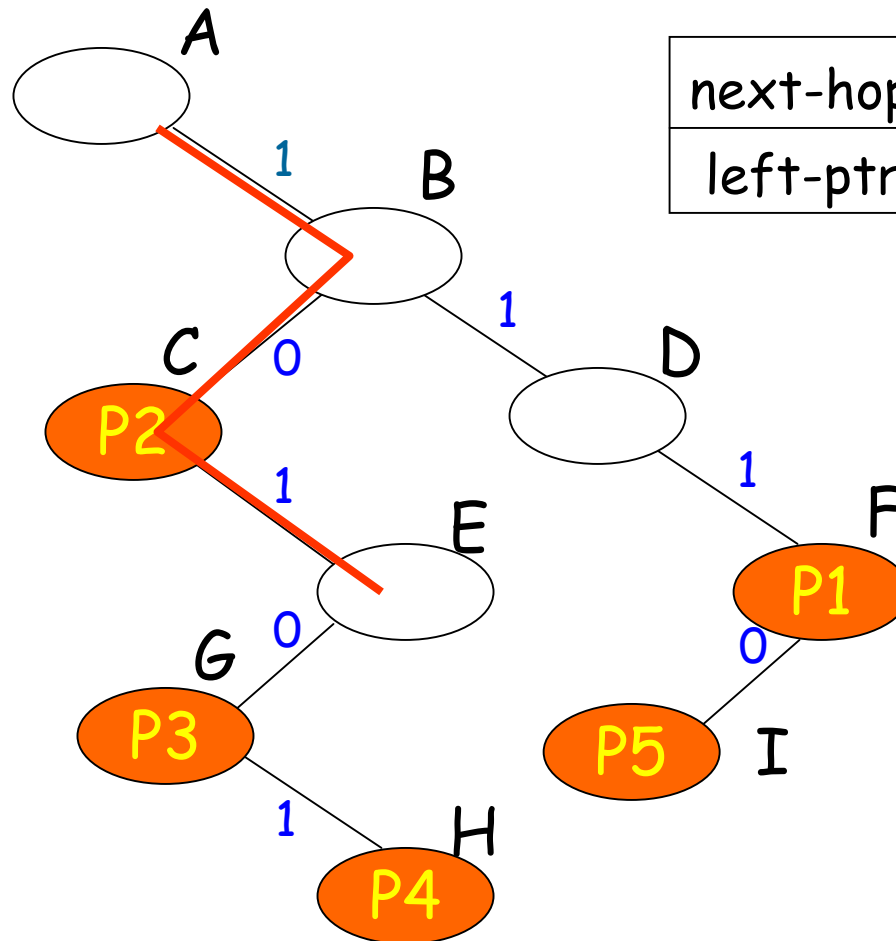
Linear Search

- Keep prefixes in a linked list
- $O(N)$ storage, $O(N)$ lookup time, $O(1)$ update complexity
- Improve average time by keeping linked list sorted in order of prefix lengths

Radix Trie

Trie node

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



Add P5=1110*

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

Lookup 10111

Radix Trie

- W -bit prefixes: $O(W)$ lookup, $O(NW)$ storage and $O(W)$ update complexity

Advantages

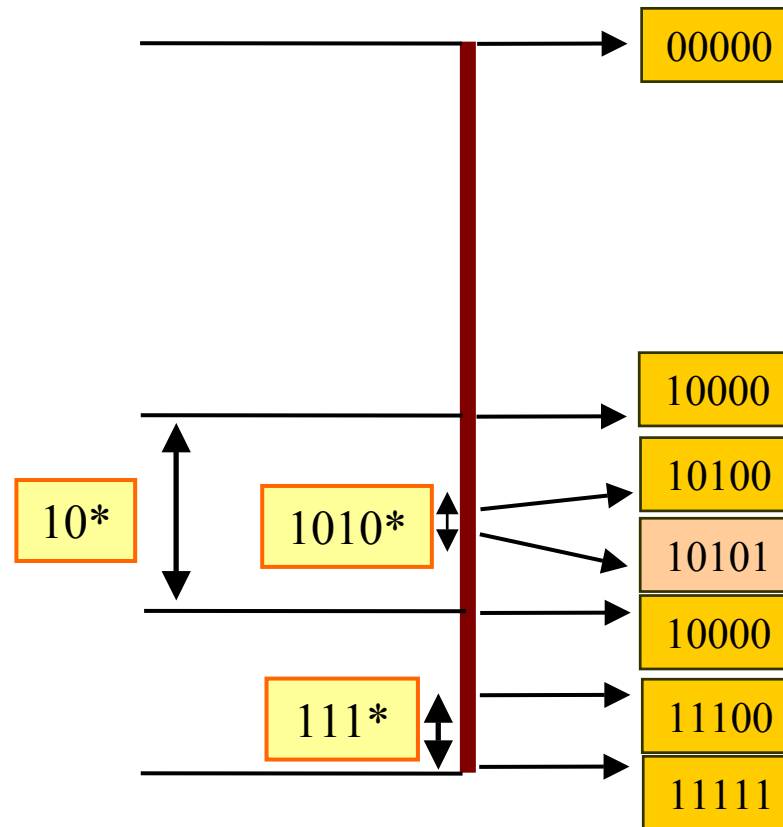
- Simplicity
- Extensible to wider fields

Disadvantages

- Worst case lookup slow
- Wastage of storage space in *chains*

Binary Search

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



Binary Search

- W -bit prefixes: $\log(2N)$ lookup, $O(N)$ storage and $O(N)$ update complexity

Advantages

- Simple
- Search time depends on N , not W

Disadvantages

- Hard for incremental update

Tuple Space Based Search

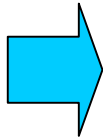
- Store a hash table for each prefix length
- Hash key is the prefix value and prefix length
- Search scheme
 - Linear search on prefix lengths
 - Binary search on prefix lengths
 - ┆ Need to provide intermediate markers
 - Guide to more specific prefix
 - ┆ Need pre-computation per marker
 - Avoid backtracking

Linear Search - An Example

Query address A = 01110011

rule Prefixes

a 0*
b 01000*
c 011*
d 1*
e 100*
f 1100*
g 1101*
h 1110*
i 1111*
j 01*
k 1100001*
p 101*



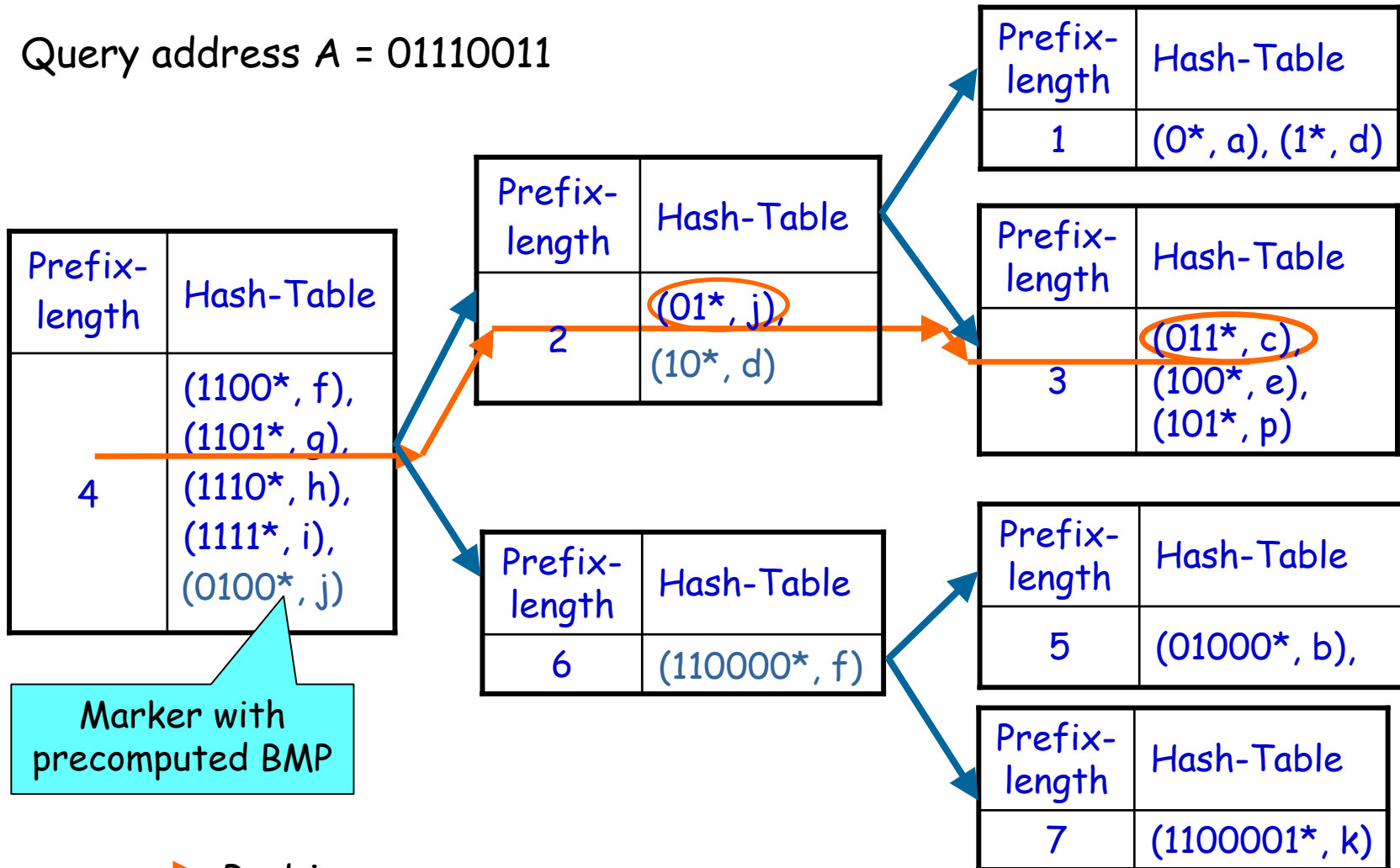
Prefix-length	Hash-Table
1	(0*, a), (1*, d)
2	(01*, j)
3	(011*, c), (100*, e), (101*, p)
4	(1100*, f), (1101*, g), (1110*, h), (1111*, i)
5	(01000*, b)
6	
7	(1100001*, k)

Probing sequence

■ The longest matching prefix for address A(=0111011) is rule c(=011*)

Binary Search - An Example

Query address A = 01110011



Marker with precomputed BMP

→ Probing sequence

Worst-case Performance

■ Linear search

- Search time: W hash probes with maximum prefix length of W bits
- Storage requirement: $O(N)$ for N prefixes

■ Binary search

- Search time: $\log W$ hash probes
- Storage requirement: $O(M \log W)$
 - A prefix may place as many as $\log W$ markers

Leaky Bucket Algorithm

Leaky bucket

- Consider ATM networks
- Bucket size is $I+L$
- A cell brings in I units of water
- Water leaks at a constant rate
- A cell is non-conforming if bucket occupancy is more than L upon its arrival

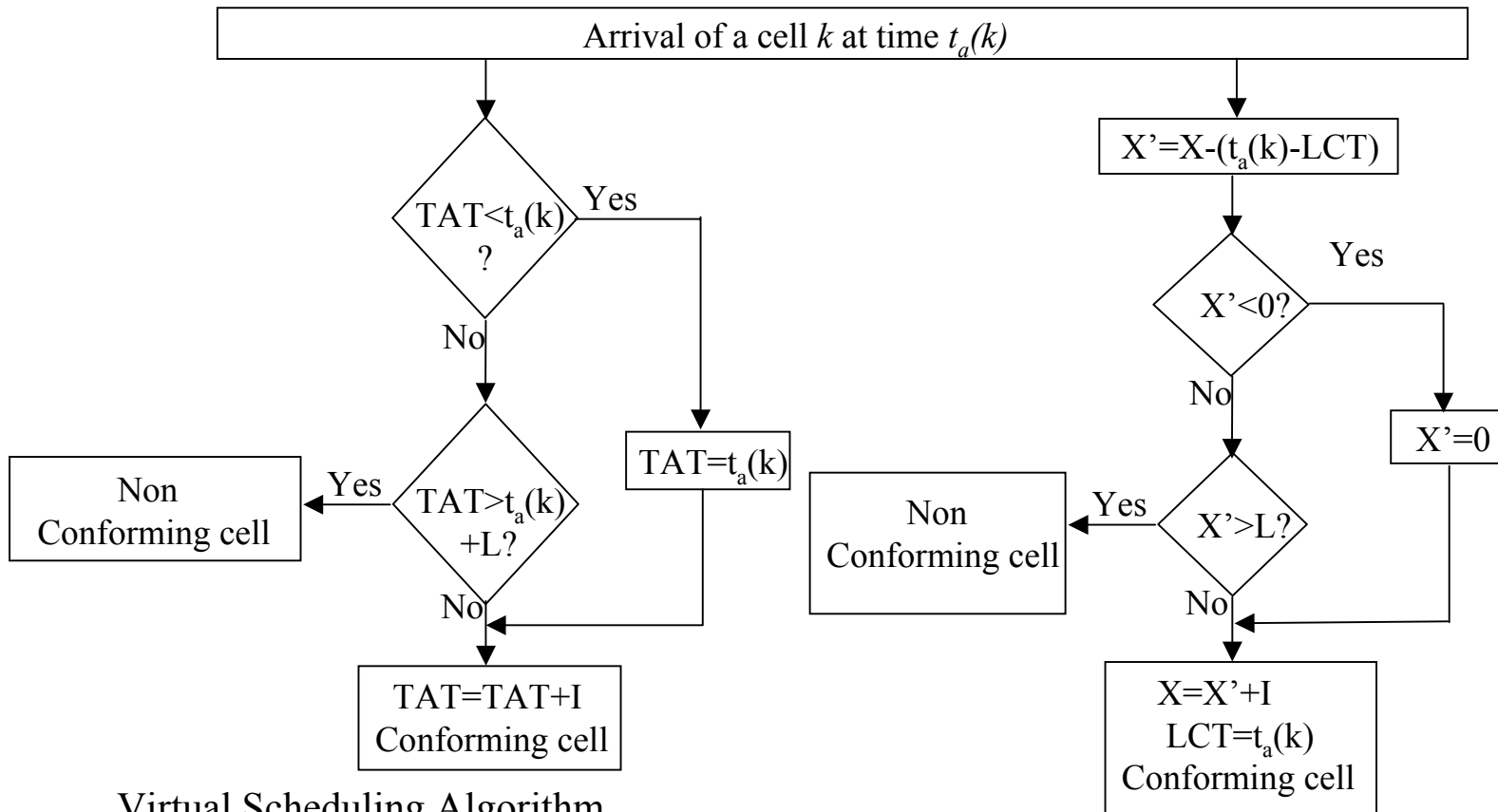
Token bucket

- Consider ATM networks again
- Bucket size is $L/I + 1$
- Tokens are generated with a constant rate $1/I$
- A new token is lost if the token bucket is full
- A conforming cell will remove 1 token
- A cell is non-conforming if there is no token upon its arrival
- Equivalent to Leaky bucket

Generic Cell Rate Algorithm (GCRA)

- Used to define conformance with respect to the traffic contract
 - Define the relationship b.w. PCR and the CDVT, and the relationship b.w. SCR and the BT
- Be a virtual scheduling algorithm or a continuous-state leaky bucket algorithm
- Be defined with two parameters: the Increment (I) and the Limit (L) \Rightarrow GCRA(I, L)

GCRA(I,L)



Virtual Scheduling Algorithm

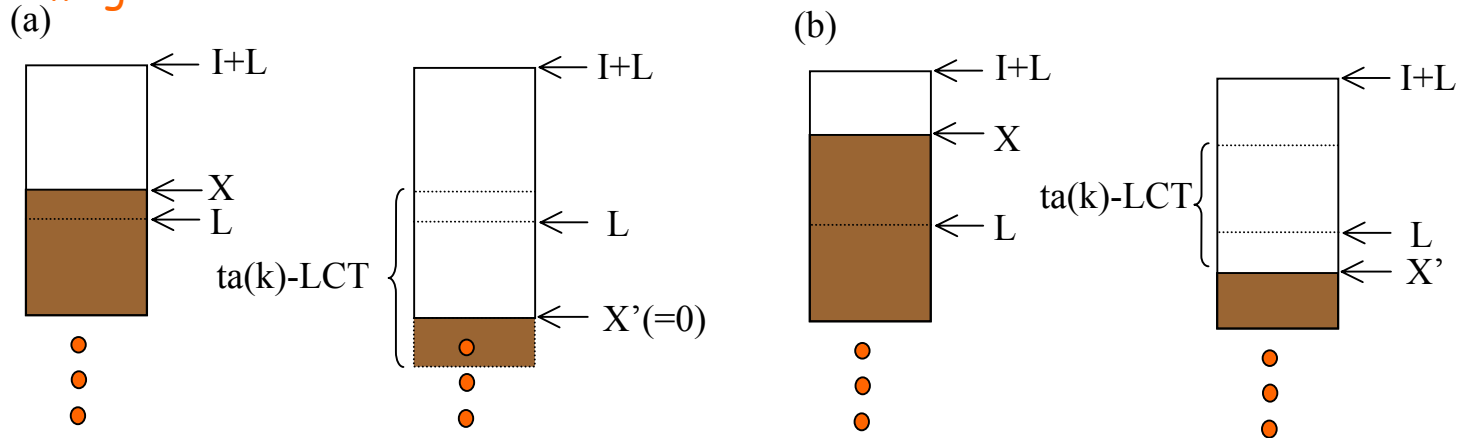
TAT: Theoretical Arrival Time
 $t_a(k)$: Time of arrival of a cell

Continuous-state Leaky Bucket Algorithm

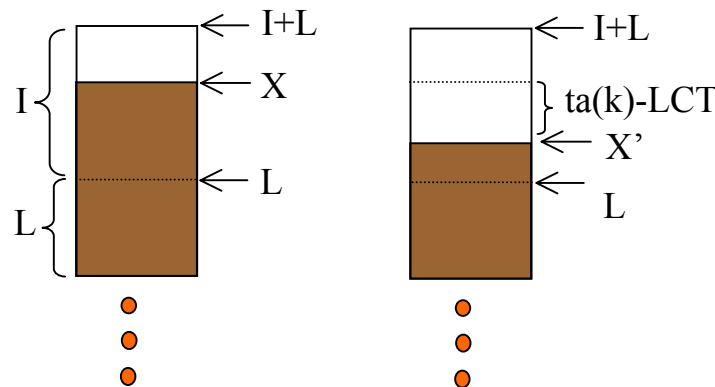
X : Value of the Leaky Bucket counter
 X' : Auxiliary variable
 LCT: Last Compliance Time

Continuous-state leaky bucket algorithm

Conforming cell



Non-conforming cell



* At the time of arrival of the first cell of the connection, $X=0$ and $LCT=t_a(k)$

Implementation of Leaky Bucket

- Store the two parameters L and I
- Store Last Conforming Time T and Bucket Occupancy X
- Upon arrival of a cell
 - Calculate updated Bucket Occupancy X'
 - Compare X' with L for conformance
 - Increment Bucket Occupancy if conformance

Singe Rate Three Color

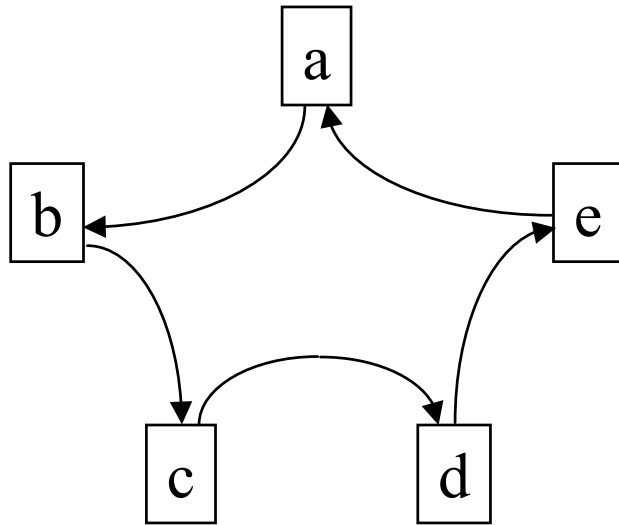
- Three parameters: CIR, CBS, EBS
- A packet is GREEN if Committed Bucket has sufficiently many tokens upon its arrival
- A packet is YELLOW if Committed Bucket does not have sufficiently many tokens, but Excess Bucket does
- A packet is RED if neither Committed Bucket nor Excess Bucket has sufficiently many tokens
- New tokens are generated with rate CIR and put in Committed Bucket, Excess Bucket if Committed Bucket is full, or lost if both buckets are full

Two Rate Three Color

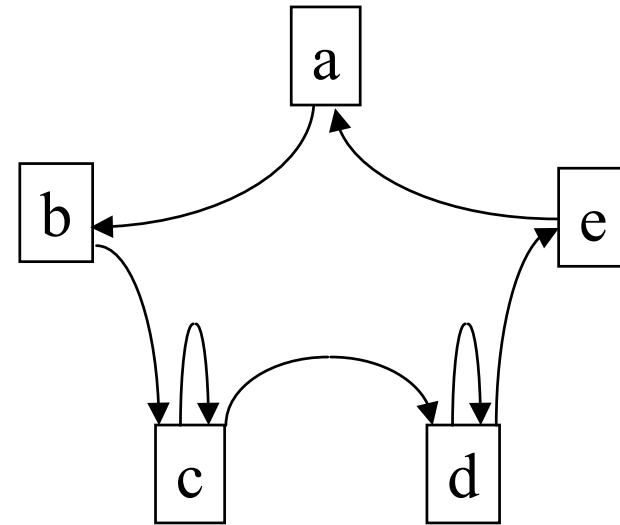
- Four parameters: CIR, PIR, CBS, PBS
- A packet is GREEN if both Committed Bucket and Peak Bucket have sufficiently many tokens upon its arrival
- A packet is YELLOW if Committed Bucket does not have sufficiently many tokens, but Peak Bucket does
- A packet is RED if Peak Bucket does not sufficiently many tokens
- New tokens are put in Committed Bucket and Peak Bucket with rate CIR and PIR, respectively

Weighted Fair Queueing

Round-robin



(a) round-robin



(b) weight round-robin

problems

- packets lengths are not the same → not fair
- Does not consider packet arrival time

Generalized Processor Sharing (GPS)

■ assumptions

- traffic is infinitely divisible
- serve multiple sessions simultaneously based on the ratio of pre-defined weights

■ properties

- fairness
- minimum throughput guarantee
- analyzable with leaky bucket admission control

■ notations:

$\phi_1, \phi_2, \dots, \phi_N$: service weight

$S_i(\tau, t)$: amount of service received by session i
during $(\tau, t]$

r : service rate of server

backlogged : a session is called backlogged if there are
packets of that session waiting for service

$Q_i(t, t)$: the session i backlog

Packet-by-Packet GPS

- PGPS (*packet-by-packet* scheme) : an approximation to GPS for operating in real world
 - ideally, PGPS wants to serve packets in the increasing departure order of GPS

■ implementation of PGPS (virtual time)

- use hypothetical ideal fluid flow model as reference
- use virtual time, $v(t)$, to represent the progress of work in the reference system.

$$V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{j \in B_j} \phi_j}, \tau \leq t_j - t_{j-1}$$

■ PGPS algorithm

Kth session, ith packet, arrival time a_i^k , length L_i^k

virtual finishing time: F_i^k

$$F_i^k = \max\{F_{i-1}^k, V(a_i^k)\} + \frac{L_i^k}{\phi_i}$$

- virtual time finishing times can be determined at the packet arrival time
- packets served in order of virtual time finishing time

Summary

- Features vs. Complexity
- CPU vs. Hardware Accelerator
- Co-processor vs. Data Path (In-Line) Accelerator
- Scalability