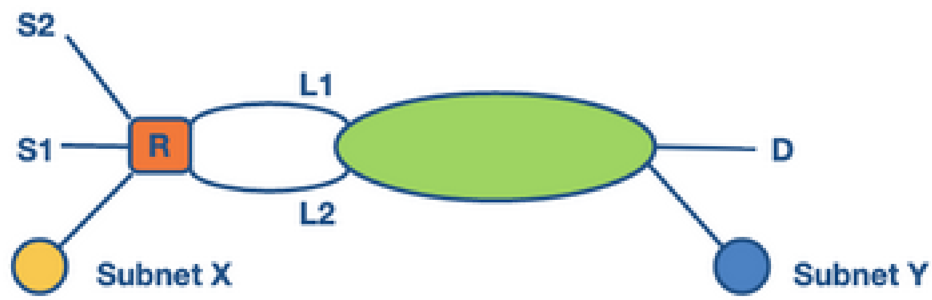
Router Design and Algorithms (Part 2)

Why We Need Packet Classification?

1. Why do we need packet classification?
    - As the Internet grows in complexity, networks require quality of service and security guarantees for their traffic
        - Longest prefix matching packet forwarding based on destination IP address is insufficient
        - Need multiple criteria, such as TCP flags, source addresses, etc.
2. Variants of packet classification
    - Firewalls: Routers implement firewalls at the entry and exit points of the network to filter out unwanted traffic or to enforce other security policies
    - Resource reservation protocols: DiffServ has been used to reserve bandwidth between a source and destination
    - Routing based on traffic type: Avoid delays for time-sensitive applications



Database at Router R

To	From	Traffic Type	Forwarding Directive
D	S1	Video	Forward via L1
*	S2	*	Drop all traffic
Y	X	*	Reserve 50 Mbps

Packet Classification

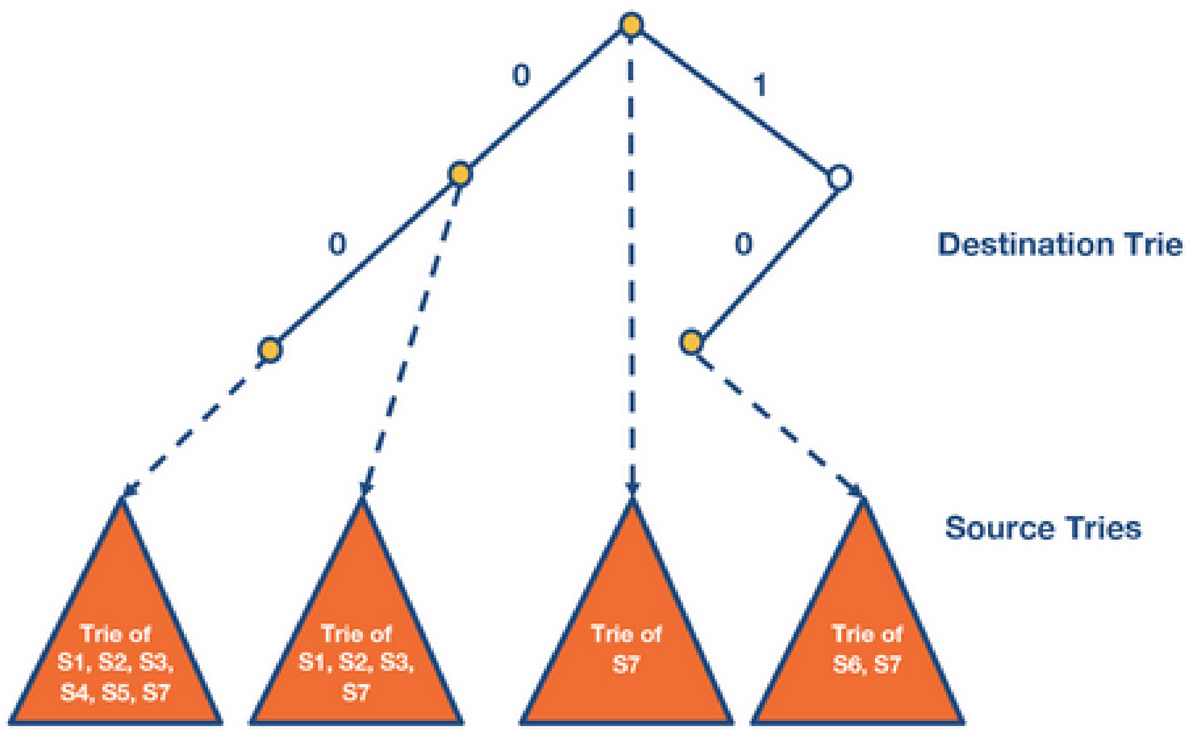
Packet Classification: Simple Solutions

1. Linear search
    - Firewall implementations perform a linear search of the rules database and keep track of the best-match rule
    - Prohibitive if there are thousands of rules to search through
2. Caching
    - Cache the results so future searches can run faster
        - Cache-hit rate can be high (80-90%) but still need to perform searches on misses
        - Even with a 90% hit rate cache, a slow linear search of the rule space will perform slowly
3. Passing labels
    - Multiprotocol Label Switching (MPLS) and DiffServ use this technology
    - MPLS is useful for traffic engineering

- Router does packet classification at the edge and maps the web traffic into an MPLS header
- Intermediate routers can apply the label without having to reclassify the packet

Fast Searching Using Set-Pruning Tries

1. Set-Pruning Tries
    - Assume a two-dimensional rule, i.e., classify packets according to both source and destination IP address
    - Naive solution: Have a trie to perform longest prefix matching on the destination IP address
        - Each leaf is a trie for performing longest prefix matching on the source IP address
2. Problems
    - A source prefix might need to be included at multiple destination prefixes
    - This can lead to memory explosion as we need to store a large amount of data, especially as the dimensionality of the rules grows



Set-Pruning Trie

Reducing Memory Using Backtracking

1. How do we use less memory for a set-pruning trie?
    - Set-pruning has high cost in memory to reduce time
        - Opposite approach: Pay in time to reduce memory
2. Backtracking approach
    - Each destination prefix D points to a source trie that stores the rules whose destination field is exactly D
    - Search algorithm then performs a "backtracking" search on the source tries associated with all ancestors of D

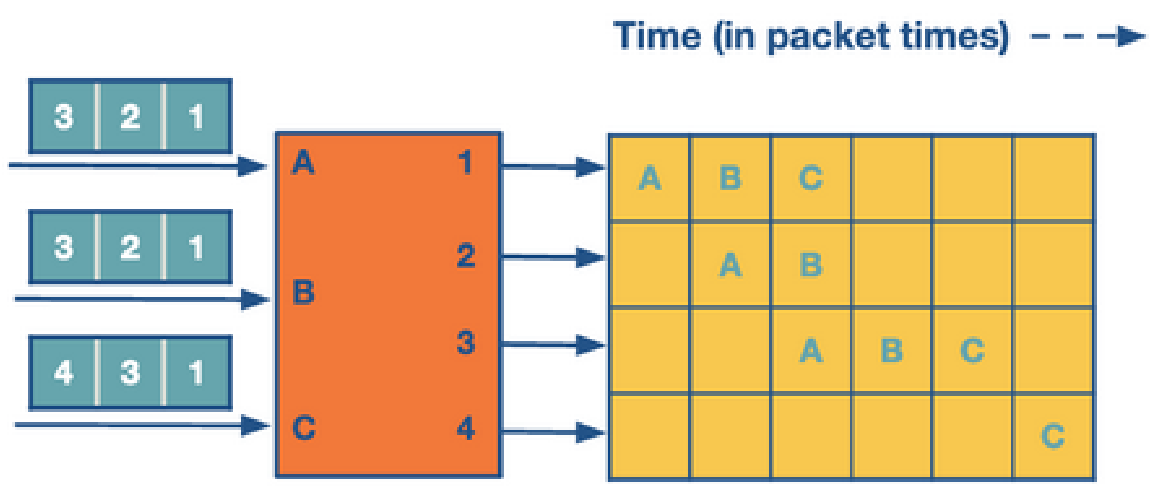
- First traverse the destination trie and find the longest destination prefix D matching the header
- Work back up the destination trie and search the source trie associated with every ancestor prefix of D that points to a nonempty source trie
- Lower memory requirements, but longer search time

Grid of Tries

1. Grid of tries approach
   - Goal: Reduce the time in the backtracking search by using precomputation
   - When there is a failure point in a source trie, precompute a switch pointer
   - Switch pointers take us directly to the next possible source trie containing a matching rule
     - Allow us to take shortcuts
     - Avoid backtracking to find an ancestor node and then traversing the source trie
     - Match the source and keep track of our current best source match, but skip source tries with source fields that are shorter than our current source match

Scheduling and Head of Line Blocking

1. Scheduling
   - Assume we have an NxN crossbar switch with N input lines, N output lines, and N^2 crosspoints
     - Each crosspoint needs to be controlled (on/off) and we need to make sure that each input link is connected with at most one output link
     - Maximize the number of input/output link pairs that communicate in parallel for better performance
2. Take-a-Ticket Algorithm
   - Each output line maintains a distributed queue for all input lines that want to send packets to it
   - When an input line intends to send a packet to a specific output line, it requests a ticket
   - Input line waits for the ticket to be served
   - Then, the input line connects to the output line, the crosspoint is turned on, and the input line sends the packet
   - Head-of-line blocking: Entire queue is blocked by the progress of the head of the queue



Head-of-line Blocking Caused by Take-a-Ticket

Avoid Head of Line Blocking

1. Output Queueing
   - Assuming that a packet arrives at an output link, it can only block packets sent to the same output link
     - Requires the fabric to run N times faster than the input links
   - Knockout scheme
     - Relies on breaking up packets into fixed sizes (cell)
     - Suppose that the same output rarely receives N cells and the expected number is k ($k < N$)
     - Can have the fabric running k times as fast as an input link instead of N
     - Primitive switching element that randomly picks the chosen output:
       * $k = 1$ and $N = 2$: Randomly pick the output that is chosen; in this case, the switching element is called a concentrator
       * $k = 1$ and $N > 2$: One output is chosen out of N possible outputs; use multiple 2x2 concentrators
       * k needs to be chosen out of N possible cells, with k and N arbitrary values. We create k knockout trees to calculate the first k winners
     - Drawback: Complex to implement
2. Parallel Iterative Matching
   - Still allow queueing for the input lines, but in a way that avoids the head-of-line blocking
   - Schedule both the head of the queue and more packets so that the queue makes progress in case head is blocked
   - Suppose we have a single queue at an input line
     - Break down the single queue into virtual queues with one virtual queue per output link
   - Algorithm runs in three rounds
     - Request phase: All inputs send requests in parallel to all outputs they want to connect with
     - Grant phase: Outputs that receive multiple requests pick a random input
     - Accept phase: Inputs that receive multiple grants randomly pick an output to send to
   - More efficient than take-a-ticket

Scheduling Introduction

1. Introduction
   - Busy routers rely on scheduling to handle routing updates, management queries, and data packets
   - Scheduling is done in real time; need to be done quickly due to increasing link speeds
2. FIFO with tail drop
   - Packets enter a router on input links
   - Looked up using the address lookup component, which gives the router an output link number
   - Switching system within the router places the packet in the corresponding output port
     - This port is a FIFO queue
   - If the output link is completely full, incoming packets to the tail of the queue are dropped
     - Potential loss in important data packets
3. Need for Quality of Service (QoS)
   - Other methods for packet scheduling: Priority, round-robin, etc
     - Useful in providing QoS guarantees to a flow of packets on measures such as delay and bandwidth
     - Flow: Stream of packets that travels the same route from source to destination and requires the same level of service at each intermediate router and gateway
     - Flows must be identifiable using fields in the packet headers
   - Router support for congestion
     - Congestion is increasingly possible as the usage has increased faster than the link speeds
     - TCP has its own ways to handle congestion, but additional router support can improve the throughput of sources by handling congestion
   - Fair sharing of links among competing flows
     - During periods of backup, packets tend to flood the buffers at an output link

    - If we use FIFO with tail drop, this blocks other flows, resulting in important connections on the clients' end freezing
  - Providing QoS guarantees to flows
    - Enable fair sharing by guaranteeing certain bandwidths to a flow
    - Guarantee the delay through a router for a flow
      * Important for video; without a bound on delays, live video streaming will not work well

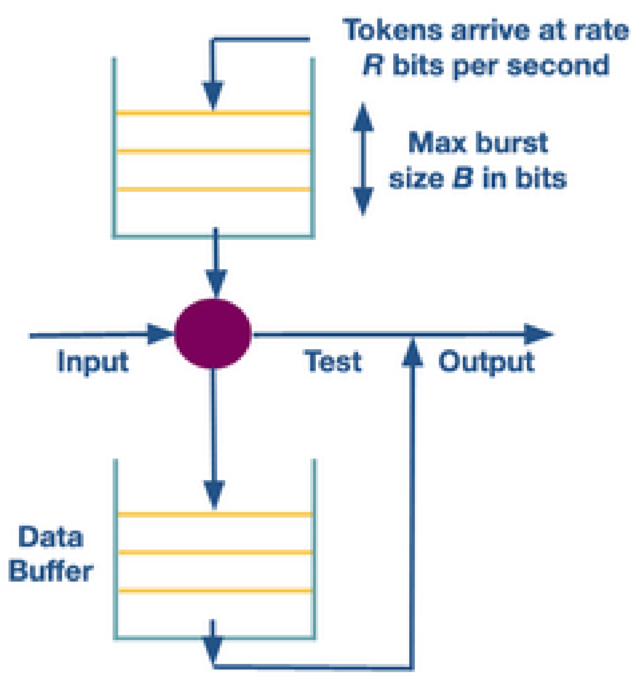
Deficit Round Robin

1. Bit-by-bit Round Robin
   - In a single round, one bit from each active flow is transmitted in a round-robin manner
     - Ensures fairness in bandwidth allocation
     - Not possible to split up packets in the real world
   - Let R(t) be the current round number at time t
     - Router can send u bits per second
     - Number of active flows is N
     - Rate of increase in round number: dR/dt = u / N
   - Round number at which packet reaches the head: S(i) = max(R(t), F(i-1))
     - R(t) is current round number
     - F(i-1) is the round at which the packet ahead of it finishes
   - F(i) = S(i) + p(i)
     - p(i) is the size of the ith packet in the flow
   - Using the above equations, the finish round of every packet in a queue can be calculated
2. Packet-level Fair Queueing
   - Emulates the bit-by-bit fair queueing by sending the packet with the smallest finishing round number
   - Packet chosen to be sent out is garnered from the previous round of the algorithm
   - Packet which had been starved the most while sending out the previous packet from any queue is chosen
   - This method provides fairness but introduces new complexities
     - Need to keep track of the finishing time at which the head packet of each queue would depart and choose the earliest one
     - Requires a priority queue implementation, which as a logarithmic time complexity in number of flows
     - If a new queue becomes active, all timestamps may have to change, which is linear in the number of flows
   - Difficult to implement at gigabit speeds
3. Deficit Round Robin
   - Bit-by-bit round robin gives bandwidth and delay guarantees, but the time complexity is too high
     - Several applications benefit only by providing bandwidth guarantees
   - Assign a quantum size Qi and deficit counter Dk for each flow
     - Qi determines the share of the bandwidth allocated to that flow
     - For each turn of the round robin, algorithm will serve as many packets in the flow i with size less than (Qi + Di)
     - If packets remain in the queue, it will store the remaining bandwidth Di for the next run
     - If all packets are serviced, Di is set to 0

Traffic Scheduling: Token Bucket

1. Introduction
   - May want to set bandwidth guarantees for flows in the same queue without separating them
     - Limit a certain type of traffic in the network to no more than X Mbps without putting this traffic into a separate queue
   - Token bucket shaping limits the burstiness of a flow by:

    - Limiting the average rate
    - Limiting the maximum burst size
2. Token Bucket Policing
    - Bucket shaping technique assumes a bucket per flow that fills with tokens with a rate of R per second and a capacity of B tokens
    - When a packet arrives, it can go through if there are enough tokens (equal to the size of the packet in bits)
    - If not, the packet needs to wait until enough tokens are in the bucket
        - Given the max size of B, a burst is limited to B bits per second
    - Implemented using a counter (can't go more than max value B and gets decremented when a bit arrives) and a timer (to increment the counter at a rate R)
    - Problem: Need one queue per flow
    - Use a modified version of the token bucket shaper to maintain one queue caled token bucket policing
        - If a packet arrives and there are no tokens in the bucket, it is dropped

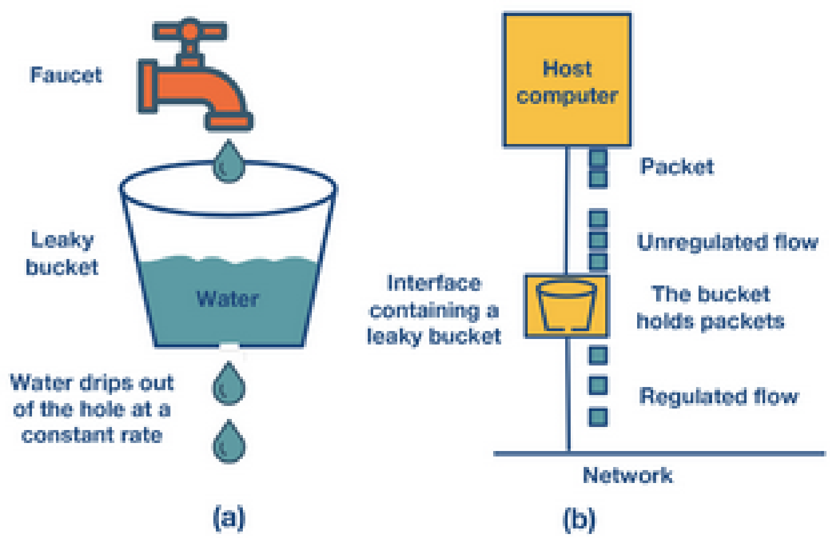


Token Bucket Policing

Traffic Scheduling: Leaky Bucket

1. Difference between policing and shaping
    - Policer: When the traffic reaches the maximum configured rate, excess traffic is dropped, or the packet's setting or "marking" is changed
        - Output rate appears as a saw-toothed wave
    - Shaper: Shaper typically retains excess packets in a queue or buffer that is scheduled for later transmission
        - Result is that excess traffic is delayed instead of dropped
        - Flow is shaped or smoothed when the data rate is higher than the configured rate
        - Traffic shaping and policing can work in tandem
2. Leaky Bucket
    - Analogous to water flowing into a leaky bucket, with the water leaking at a constant rate

  - Bucket (capacity B) represents a buffer that holds packets
  - Water corresponds to incoming packets
  - Leak rate, r, is the rate at which packets are allowed to enter the network, which is constant irrespective of the rate at which packets arrive
- If an arriving packet does not cause an overflow when added to the bucket, it is conforming
  - Otherwise, it is non-conforming
- Irrespective of the input rate of the packets, the output rate is constant
  - Leads to uniform distribution of packets sent to the network



Leaky Bucket