Routers implementations

Switching Technology S38.165

http://www.netlab.hut.fi/opetus/s38165

Router implementations

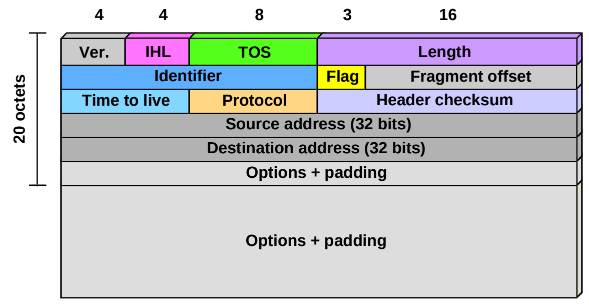
- General of routers
- Functions of an IP router
- Router architectures
- Introduction to routing table lookup

General of routers

- Router is a network equipment, which
  - performs packet switching operations
  - operates at network layer of OSI protocol reference model
  - switches/routes variable length packets
  - routing decision based on address information carried in packets
- Router is used to connect two or more networks that may or may not be similar
- Routers communicate with each other by means of routing messages to
  - exchange routing information
  - resolve next hop addresses
  - maintain network topology to make routing decisions

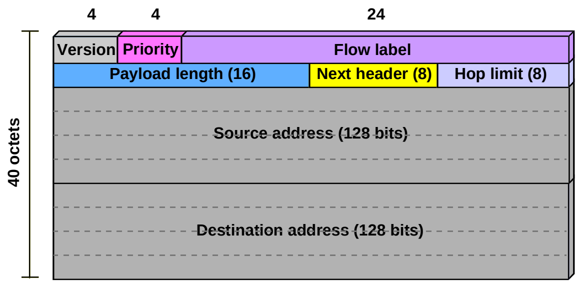
IPv4 packet structure

Version - IP version number
IHL - Internet header length in 32-bit words (min. size = 5)
TOS - type of service (guidance to end-system IP modules and router along transport path)
Length - total length (header + payload) of IP packet in octets
Identifier - sequence number, which together with address and protocol fields identify each IP packet uniquely



Flag - used in connection with fragmentation: **more bit** indicates whether this fragment is the last one of a fragmented packet and **don't fragment bit** inhibits/prohibits packet fragmentation
Fragment offset - indicates where in the original user data message this fragment belongs. measured in 64-bit units (all but last fragment has a data field that contains a multiple of 64-bit payload)
Time-to-live - defines the maximum time in seconds a packet can be in transit across the Internet (decremented by each visited router by a defined amount)
Protocol - indicates the type of protocol (TCP, UDP, etc.) carried by the IP packet
Header checksum - carries a checksum calculated over the header bits

IPv6 packet structure

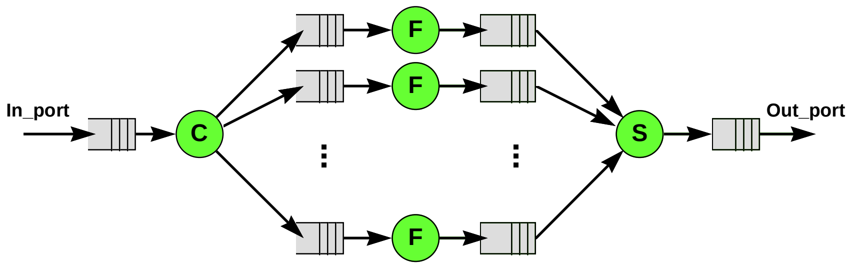


Version - version number of IP protocol
Priority - priority of the packet
Flow label - indicates the type payload carried by the packet
Payload length - indicates the number of octets in the payload field
Next header - indicates the type of additional (extension) header following the main header
Hop limit - value for the maximum number of hops the packet is allowed to travel in a network

Router implementations

- General of routing
- **Functions of an IP router**
- Router architectures
- Introduction to routing table lookup

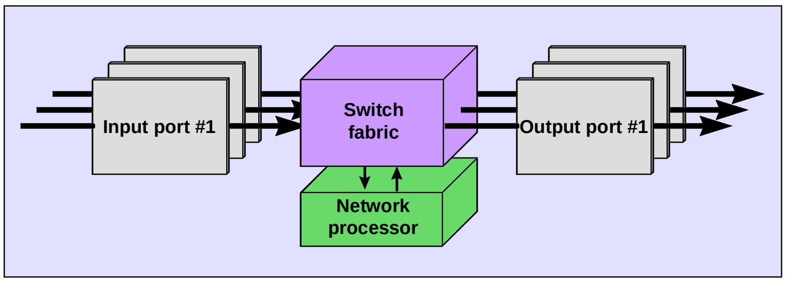
Major tasks of a router



C - Classify (classification, filtering and routing)
F - Forward (transfer of packets from input interfaces to addressed output interfaces)
S - Scheduling (transmission of data packets based, e.g. on priority)

Main functional blocks of a router

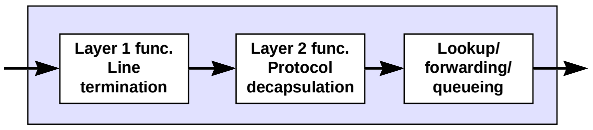
Generic router architecture



Input port functionality

- Layer 1 termination of incoming physical links (e.g. SDH, Ethernet)
- Layer 2 frame decapsulation to inter-operate with data-link protocols of connected networks
- Forwarding of control packets, e.g. routing information packets (RIP, OSPF, IGMP), to network processor to update routing table and topology
- Some implementations distribute a copy of routing table and table lookup to each input port, while some other implementations forward all incoming packets to a centralized routing processor

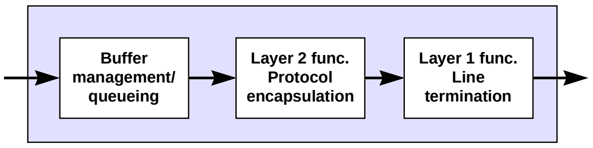
Input port functionality


Layer 1 func. Line termination
Layer 2 func. Protocol decapsulation
Lookup/ forwarding/ queueing

Output port functionality

- Buffering of outbound packets
- Scheduling of buffered packets to guarantee required QoS
- Layer 2 frame generation and encapsulation of packets into frames (e.g. AAL5/ATM/SDH, PPP/SDH and Ethernet)
- Layer 1 physical signal generation

Output port functionality


Buffer management/ queueing
Layer 2 func. Protocol encapsulation
Layer 1 func. Line termination

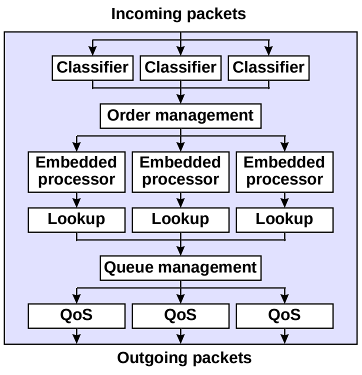
Switch fabric functionality

- Main function is to route data packets from input ports to addressed output ports
- Depending on the switch fabric implementation, packets are transported through the fabric either as uniform variable length packets or they are fragmented to fixed size data units
- In either case, extra information is added in front of the packets to direct them through the fabric
  - switching of whole packets is usually applied in low-speed routers
  - switching of fragments is normally used in high-speed routers
- Majority of switch fabrics are based on three basic architectures: bus based, memory based and interconnection network based

Network processor functionality

- Maintenance of routing table
- Execution of routing protocols
- Maintenance of routing topology
- Performance of network management
- Wire-speed operation obtained by implementing key functions in hardware
- Processing of packets
  - classification
  - order management
  - acceleration of lookup
  - queue management
  - QoS engine

Network processor functionality



Router classification

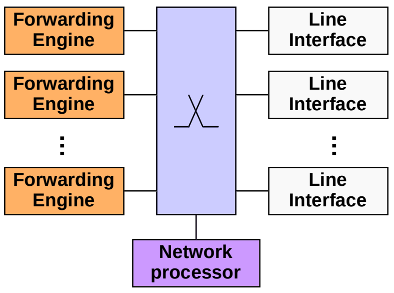
- **Access routers**
  - link homes and small business to ISPs (Internet Service Provider)
  - need to support a variety of access technologies, e.g. high-speed modems, cable modems and ADSL
- **Enterprise/metropolitan routers**
  - used as campus and office interconnects
  - QoS guarantees for local traffic
  - support of several network layer protocols (e.g. IP and IPX)
  - support of additional features, such as firewalls, security policies and virtual LANs
- **Backbone/long haul routers**
  - interconnect enterprise routers
  - huge number of packets per second => very high speed requirement
  - critical components for interworking => reliability of utmost concern

Router implementations

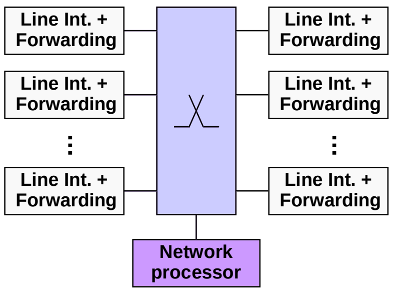
- General of routing
- Functions of an IP router
- **Router architectures**
- Introduction to routing table lookup

Basic types of router architecture

Router with forwarding engines



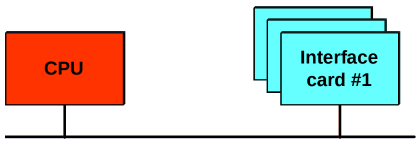
Router with added processing power in interfaces



First generation router architecture

- Network layer protocols were constantly changing
  => adaptable solution was needed
  => a single and common purpose processor structure was a reasonable one in which operating system in central role
- Low throughput (packets transferred twice through the bus) did not scale well with increasing line speeds

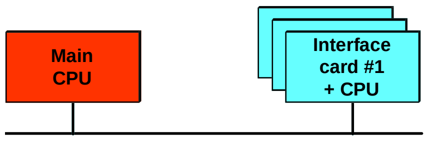
Shared bus, a single processor card and line interface cards



Second generation router architecture

- Each line card implemented a processor
  => distributed and parallel routing became available
- Main processing unit took care of delivery of routing information to line interface cards
- Operating system still in central role
- Cache memories were introduced to speed up routing decisions (most recently used routing entries kept in cache)
- Increased throughput, but shared bus still a bottleneck
- Solution did not scale with increasing line speeds

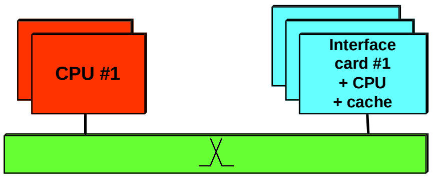
Shared bus and a processor on each line interface card



Third generation router architecture

- Shared bus replaced with more powerful switch fabrics (e.g. multi-stage and crossbar)
- Parallel processing units (based on general purpose processors)
- Cache memories to enhance routing decision making
- Operating system still played an important role
- Communication between line interfaces no more a problem
- QoS increases processing power requirement (IP/TCP/application)
- Did not scale well enough with the most advanced line speeds

Switch fabric and more processing power

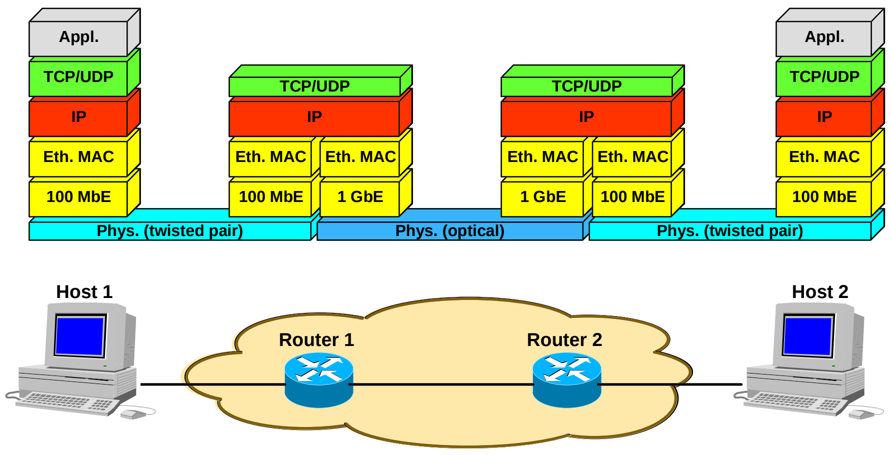


Support of differentiated services

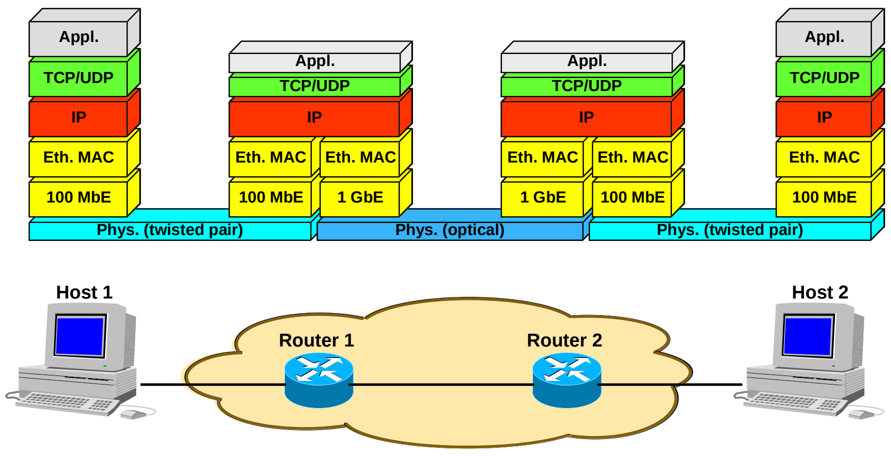
Traditional routers are limited in terms of their quality of service and differentiation features. Advances in research and hardware capabilities have provided mechanisms to overcome these limitations. Following operations, possible today to carry out in high speed, allow provisioning of differentiated services:

- **Packet classification** - distinguish packets and group them according to their different requirements
- **Buffer management** - determine how much buffer space should be given to certain kinds of network traffic and which packets should be discarded in case of congestion
- **Packet scheduling** - decide that the packet servicing order meets the bandwidth and delay requirements of different types of traffic

DiffServ routing

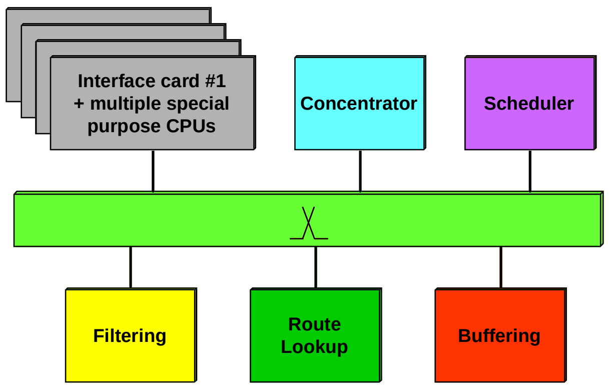


DiffServ routing (cont.)



Sharing of processing resources and pipelining

DiffServ-optimized router architecture



Sharing of processing resources and pipelining (cont.)

- Packet processing divided into a number of consecutive processes - each process has a dedicated processing unit (buffering, filtering, routing, etc.)
- Pipelined processes shared by several interfaces to increase number of line interfaces - concentrator schedules packets for processes
- QoS-based scheduler takes care of packet transmission from buffers to outbound interfaces

Packet processing capacity

- Packet processing capacity of a router is given as number of forwarded packets/second and/or forwarded bits/second
- Tasks affecting forwarding speed
  - link protocol processing delay (input and output)
  - address lookup time
  - switching of packets from input ports to outputs ports
  - queueing at output ports and possibly at input ports
- Other tasks that may have an impact on forwarding speed
  - routing table management/updates
  - network and router management
- In high capacity routers, routing table lookups are a major problem
- Queueing is the main component of routing latency
- Routing capacity requirement determined by the shortest packets

Future challenges

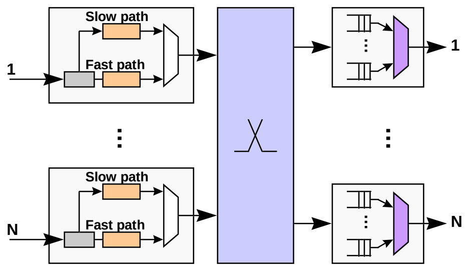
- Increase of line speeds
  - 100 Mbit/s => 1 Gbit/s => 10 Gbit/S => 40/100 Gbit/s
- QoS-support => increased processing need
  - DiffServ, IntServ, MPLS, ...
- From "best effort" service to controlled use of network resources
  => programmable network nodes
- Different needs in the core and edge networks
  - huge routing capacity in the core network (> 10 million packets/s)
  - a lot of functionality and intelligence in the edge routers

Speedup mechanisms for routing table lookup

- Caching
  - routing table entries of most lately arrived packets or entries most frequently accessed are stored in cache memory
- Pipelining
  - different phases of routing table lookup are executed by different pippelined processing units
- Distribution of lookup to interfaces or to several routing engines
  - network processor takes care of routing table updates and distributes updated tables to separate interface/routing engines
- In centralized routing solutions only packet headers are sent to a routing processor
- Implementation of lookup functionality in hardware (at the expense of flexibility)

Caching to speedup packet processing and forwarding

- When a packet with a new destination address arrives to an input port, it passes through the conventional routing process (slow path) and its routing entry is stored in cache memory
- Subsequent packets carrying the same destination address are routed using the routing entry in the cache memory (fast path)
- A routing entry is removed from cache when predefined conditions to keep it in cache expire, e.g. packet arrival rate declines or time between successive packets becomes too long



High speed router examples

- **GSR /Cisco**
  - first gigabit router on the market
  - switching capacity of 27.5 Gbits/s
  - equipped with POS (Packet Over Sonet) and ATM interfaces
- **12000 Terabit System /Cisco**
  - initial switching capacity of 150 Gbits/s, but scalable up to 5 Tbits/s
  - can be equipped with OC192/STM-64 (10 Gbits/s) interfaces
- **NX64000 /Lucent (Nexabit)**
  - one of the highest capacity routers (6.4 Tbits/s)
  - supports interface rates up to OC192/STM-64
  - distributed programmable hardware based forwarding engine
  - 1 million routing entries on each line card
  - 40 ms delay guarantee for variable size packets

High speed router examples (cont.)

- **MGR (Multi-Gigabit Router) /BBN Technologies**
  - forwarding rate up to 21 million packets per second
  - switching backplane capacity of 50 Gbits/s
  - multiple line cards and separate forwarding engine cards plugged into a high-speed switch
  - only packet headers are directed to forwarding engines
  - payloads queued on line cards
- **TSR (Terabit Switch-Router) / Avici**
  - designed to be scalable from 600 Mbit/s to several Tbits/s
  - hardware based routing, forwarding, multi-casting and QoS service
  - each line card implements a 70 Gbit/s router and 20 such line cards fit into a dual-shelf chassis

  => total switching capacity is 1.4 Tbits/s

Example of routing table lookup speed determination

In a distributed routing table lookup solution, each input port implements a routing table. What is the maximum allowed routing decision delay if the input link is a 100 Mbit/s Ethernet link or 1 Gbit/s link and the router should operate at wire-speed ?

Solution:

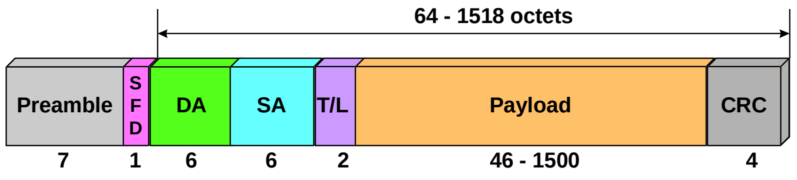
- In both example cases, the routing decision delay requirement corresponds to maximum packet arrival rate at these interfaces
- The maximum packet arrival rate is encountered when there is a constant stream of minimum size Ethernet frames
- The minimum size 100 MbE frame is 64 octets and there are 8 octets of preamble and SFD information in front of each frame and additionally there is always a 0.96 µs time gap between successive frames

Example of routing table lookup determination (cont.)

Solution (cont.):

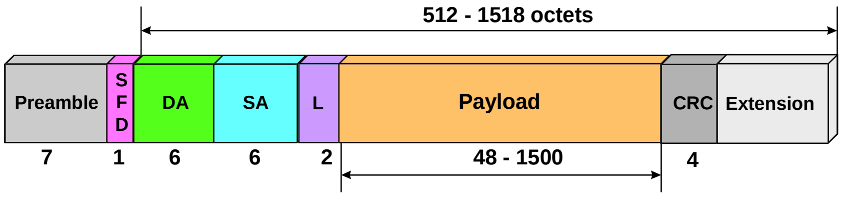
- Time required to transmit 72 octets (64+8) at the speed of 100 Mbit/s is 5.76 µs => minimum time interval between successive frames is 5.76 + 0.96 µs = 6,72 µs, which is also the maximum allowed routing decision delay for a 100 MbE input port
  => forwarding capacity is about 149 000 packets/s
- The minimum size 1 GbE frame is 512 octets and there are 8 octets of preamble and SFD information in front of each frame and there is a 96 ns time gap between successive frames
- Time required to transmit 520 octets (512+8) at the speed of 1 Gbit/s is 4.16 µs => minimum time interval between successive frames is 4.16 + 0.096 µs = 4.256 µs, which is also the maximum allowed routing decision delay for a 1 GbE input port (frame bursting excluded) => forwarding capacity is about 235 000 packets/s

10/100 MbE frame



Preamble - AA AA AA AA AA AA AA (Hex)
SFD - Start of Frame Delimiter AB (Hex)
DA - Destination Address
SA - Source Address
T/L - Type (RFC894, Ethernet) or Length (RFC1042, IEEE 802.3) indicator
CRC - Cyclic Redundancy Check
Inter-frame gap 12 octets (9,6 µs /10 MbE)

1GbE frame



Preamble - AA AA AA AA AA AA AA (Hex)
SFD - Start of Frame Delimiter AB (Hex)
DA - Destination Address
SA - Source Address
T/L - Type (RFC894, Ethernet) or Length (RFC1042, IEEE 802.3) indicator
CRC - Cyclic Redundancy Check
Inter-frame gap 12 octets (96 ns /1 GbE)
Extension - for padding short frames to be 512 octets long

Router implementations

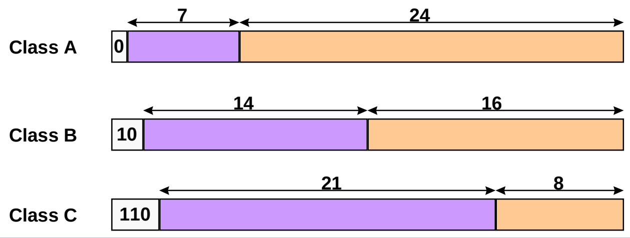
- General of routing
- Functions of an IP router
- Router architectures
- **Introduction to routing table lookup**

Classful addressing scheme

- In IPv4 addresses are 32 bits long - broken up into 4 groups of 8 bits and represented usually as four decimal numbers separated by dots, e.g., 10000010 01010110 00010000 01000010 = 130.86.16.66
- IP intended for interconnecting networks => routing based on network is a natural choice (rather than based on host)
- IP address scheme initially used a simple two level hierarchy - networks at the top level and hosts at the bottom level
- Network part (i.e. address prefix) corresponds to the fist bits
- Prefixes written as bit strings up to 32 bits in IPv4 followed by "*"
  - e.g. 1000001001010110* represents all the 2^{16} addresses that begin with bit pattern 1000001001010110
  - an alternative way is to use dotted-decimal expression, i.e., 130.86/16 (number after the slash indicates length of prefix)

Classful addressing scheme

- With the two level hierarchy, IP routers forwarded packets based on the network part, until packets reached their destination network
- Forwarding table only needed to store a single entry to forward packets to all hosts attached to the same network - technique is called address aggregation and allows prefixes to represent a group of addresses
- Three different network sizes were defined: A, B and C (see figure)



Classful addresses

- Classful addressing scheme worked well in the early days of the Internet
- Two basic problems appeared when the number of hosts and networks grew
  - address space was not efficiently used (only three possible network sizes available) and was getting exhausted very rapidly
  - forwarding tables in the backbone routers grew rapidly, because routers maintain an entry in the forwarding table for every allocated network address
    => long lookup times and larger memory requirement

Classless InterDomain Routing (CIDR)

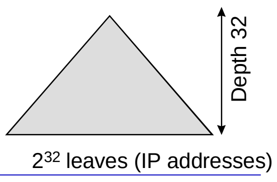
- CIDR was introduced to allow more efficient use of IP address space and to slow down growth of backbone routing tables
- CIDR allows prefixes to be of arbitrary length, not just 8, 16 or 24 as in classful address scheme
- A network that has identical routing information for all sub-nets except for a single one, requires only two entries in the routing table
- In CIDR, each IP route is represented by a `route prefix` / `prefix length` pair
  - prefix length indicates number of significant bits in a route prefix
  - e.g. a routing table may have prefixes 12.0.54.8/32, 12.0.54.0/24 and 12.0.0.0/16. If a packet is destined to address 12.0.54.2, the second prefix matches

Difficulties with longest match prefix search

- In classful addressing scheme, prefix length is coded in the most significant bits of an IP address
  => address lookup is a relatively simple operation - prefixes are organized in three separate tables (A, B and C)
  => an exact prefix match could be found using standard search algorithms based on hashing or binary search
- CIDR allows reduced size of forwarding tables, but address lookup problem becomes more complex
  => prefixes are of arbitrary length and no longer correspond to the network part
  => search in the forwarding table can no longer be performed by exact matching, because the length of the prefix cannot be derived from the address itself
  => searching in two dimensions: bit pattern value and length

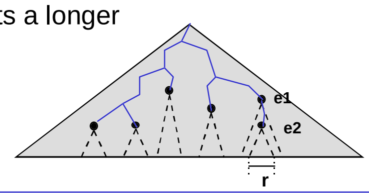
Route lookup

- Primary goal in designing a data structure to be used in a forwarding table is to minimize lookup time, i.e.
  - minimize number of memory accesses required during lookups
  - minimize size of data structure (to fit partly or entirely into a cache)
- Secondary goals of a data structure
  - as few instructions during lookup as possible
  - keep the entities naturally aligned as much as possible to avoid expensive instructions and cumbersome bit-extraction operations
- A binary tree spanning the entire IP address space has a height of 32 and number of leaves is 2^{32}



Route lookup (cont.)

- A prefix of a routing table entry defines a path in the tree ending in some point and all IP addresses (leaves) in a sub-tree, rooted at that node, should be routed according to that routing entry, i.e. each routing table entry defines a range of IP addresses with identical routing information
- If several routing entries cover an IP address, the rule of the longest match is applied, i.e. the longest applicable prefix should be used
- In the figure below, **e2** represents a longer match than **e1** for addresses in range **r**



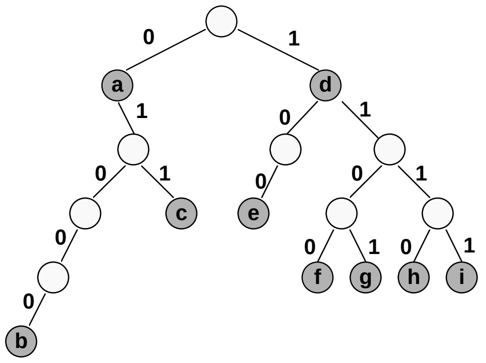
Route lookup based on binary trie

- A trie is a tree-based structure allowing to organize prefixes on a digital basis by using the bits of prefixes to direct the branching
- In a trie, a node on level **k** represents the set of all addresses that begin with the **k** bits consisting of the string of bits labeling the path from the root to that node, e.g. node **c** in the figure on the next slide is at level 3 and represents all addresses beginning with the sequence 011
- Nodes that correspond to prefixes are shown in a darker shade - these nodes contain the forwarding information or a pointer to it
- Some addresses may match several prefixes, e.g. addresses beginning with 011 will match prefixes **c** and **a** => prefix **c** is preferred because it is more specific (longest match rule)

A binary trie for a set of prefixes

<u>Prefixes</u>:

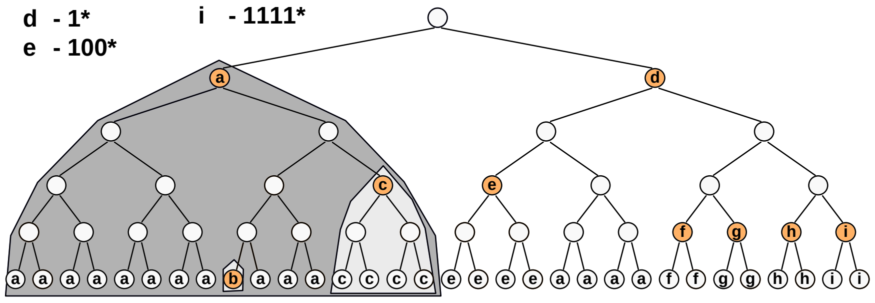
a	- 0*
b	- 01000*
c	- 011*
d	- 1*
e	- 100*
f	- 1100*
g	- 1101*
h	- 1110*
I	- 1111*



Address space of 5-bit long addresses

<u>Prefixes</u>:

a	- 0*	f	- 1100*
b	- 01000*	g	- 1101*
c	- 011*	h	- 1110*
d	- 1*	i	- 1111*
e	- 100*		



a
d
c
e
f
g
h
i
a a a a a a a a b a a a c c c c e e e e a a a a f f g g h h i i

Route lookup based on binary trie

- Tries allow finding the longest prefix that matches a given destination address and the search is guided by the bits of the destination address
- While traversing the trie and visiting a node marked as a prefix, this prefix is marked as the longest match found so far
- The search ends when no more branches to take exist and the longest match is the prefix of the latest visited prefix node
- An example address 10110
  - from root move to the right (1st bit value = 1) to node **d** marked as a prefix, i.e. 1st found prefix is 1*
  - then move to the left (2nd bit value = 0) to a node not marked as a prefix - prefix **d** still valid
  - 3rd address bit = 1, but at this point there is no branch to the right
  => **d** is the last visited prefix node and prefix of **d** is the longest match

Route lookup based on binary trie (cont.)

- Going trough a trie is a sequential prefix search by length when trying to find a better match
  - begin looking in the set of length-1 prefixes, located at level 1
  - then proceed in the set of length-2 prefixes at level 2,
  - then proceed to level 3 and so on
- While stepping through a trie, the search space reduces hierarchically
  - at each step, the set of potential prefixes reduces and the search ends when this set is reduces to one
- Update operations are straightforward
  - inserting a new prefix proceeds as a normal search and when arriving to a node with no branch to take, insert the necessary node
  - deleting a prefix proceeds also as a search and when finding the required node, unmark it as a prefix node and delete it if necessary

Path-compressed tries

- In binary tries, long sequences of one-child nodes may exist and these bits need to be inspected even though the actual branching decision has been made
  => search time can be longer than necessary
  => one-child nodes consume additional memory
- **Path-compression** technique can be used to remove unnecessary one-way branch nodes and reduce search time

Path-compressed tries (cont.)

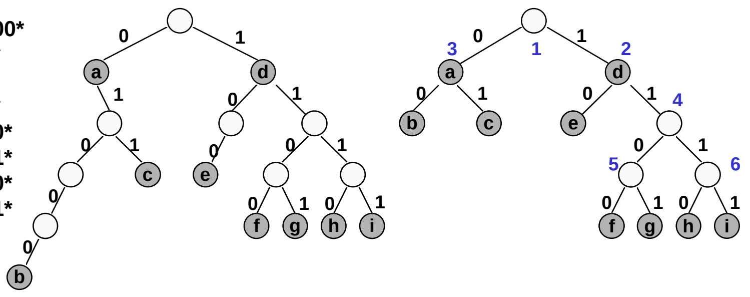
- Path-compression was first introduced in a scheme called **Patricia**, which is an improvement of the binary trie structure
  - it is based on the observation that an internal node, which does not contain a prefix and has only one child, can be removed
  - removal of internal nodes requires information of missing nodes to be added in remaining nodes so that search operation can be performed correctly, e.g. a simple mechanism is to store a number, which indicates how many nodes have been skipped (skip value) or number of the next address bit to be inspected
- There are many ways to exploit path-compression technique, an example is shown on the next slide

A path-compressed trie example

Prefixes:

a	**- 0***
b	**- 01000***
c	**- 011***
d	**- 1***
e	**- 100***
f	**- 1100***
g	**- 1101***
h	**- 1110***
I	**- 1111***

Uncompressed binary trie **Compressed binary trie**



A path-compressed trie example (cont.)

- Two nodes preceding **b** have been removed
- Since prefix **a** was located at one child node, it was moved to nearest descendant, which is not a one-child node
- If several one-child nodes, in a path to be compressed, contain prefixes, a list of prefixes must be maintained in some of the nodes
- Due to removal of one-child nodes, search jumps directly to an address bit where a significant decision is to be made
  => bit position of the next address bit to be inspected must be stored
  => bit strings of prefixes must be explicitly stored

Search in a path-compressed trie

Search goes as follows:

- Start from root and descent in the trie under the guidance of the address bits, but this time only inspect bit positions indicated by the bit position number in the nodes traversed
- When a node marked as a prefix is encountered, comparison with the actual prefix is performed - this is needed because during the descent in the trie we may skip some bits
- If a match is found, we proceed traversing the trie and keep the prefix as the best match prefix (BMP) so far
- Search ends when a leaf is encountered or a mismatch found
- BMP is the last matching prefix encountered

A path-compressed trie example (cont.)

In the previous example case, take an address begging with 010110

- start from root and since its bit position number is 1, inspect the first bit of the address
  => 1st bit is 0 => go to the left
  => since this node is marked as a prefix, compare prefix **a** (“0”) with the corresponding part of the address => they match
  => keep **a** as the BMP so far
  => bit position number of the new node is 3 so skip the 2nd address bit and inspect the 3rd one, which is 0 => proceed left
  => next node includes a prefix so compare prefix **b** with the corresponding part of address
  => no match => stop search => the last recorded BMP is **a**

New IP lookup algorithms

- Difficulty in longest prefix matching is related to its dual dimensions: length and value
- New schemes for fast address lookup differ in the dimension to search and whether this search is linear or binary
  - search on value
  - search on length
  - prefix transformation
  - compression techniques

New IP lookup algorithms (cont.)

Search on value approach

- Sequential search on values is a straightforward scheme to find BMP, but the problem is that search space is reduced only by one prefix at each step
- Data structure is an array with unordered prefixes
- Search goes through all entries (no length dimension)
  => search complexity is O(N)
  => scalability problem (binary search would be better)

Search on length approach

- Linear and binary search applied
- In linear search on length prefixes of length i searched at step i
- Prefixes organized in such a way that stepping through a trie reduces the set of possible prefixes (e.g. multibit tries do linear search, but inspect several bits at each step)

New IP lookup algorithms (cont.)

Search on length approach (cont.)

- Alternative way to organize prefixes for search on length is to use a different table for each possible prefix length - at each search step a particular table is searched (for example by hashing)

Prefix transformation

- Forwarding information is specified with prefixes that represent ranges of addresses
- Forwarding information can be expressed with different sets of prefixes => various transformations are possible, but the most common one is **prefix expansion**
- Expanding a prefix means transforming one prefix into several longer and more specific prefixes that cover the same range of addresses
- Example: prefix 1* can also be specified with prefixes 10* and 11* or with 100*, 101*, 110* and 111*

New IP lookup algorithms (cont.)

Prefix transformation (cont.)

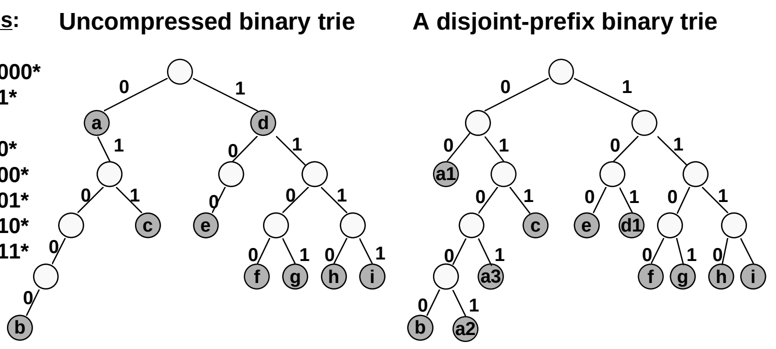
- Properly done prefix expansion can lead to a set of prefixes that have fewer different lengths
- In case of overlapping prefixes, i.e. several prefixes match, longest match rule is applied
- One way to avoid longest match rule is to transform a given set of prefixes into a disjoint set of prefixes (no overlapping), i.e. prefixes located at the leaves and not at internal nodes of a trie
- A disjoint binary trie is obtained by adding leaves to nodes that have only one child - these new leaves are new prefixes that inherit forwarding information of the closest ancestor prefix - internal nodes marked as prefixes are unmarked
- Example (leaf pushing technique): prefixes **a1**, **a2** and **a3** have inherited forwarding information of prefix **a**, which has been suppressed - **d1** obtained in a similar way (see figure/next slide)

New IP lookup algorithms (cont.)

Leaf pushing technique

<u>Prefixes</u>:

a	- 0*
b	- 01000*
c	- 011*
d	- 1*
e	- 100*
f	- 1100*
g	- 1101*
h	- 1110*
I	- 1111*



New IP lookup algorithms (cont.)

Compression techniques

- Compression attempts to remove redundancy from encoding
- Compression idea comes from the fact that expanding the prefixes increases redundancy
- Compression should reduce memory consumption
  => retrieving information from the compressed structure is done easily and with the minimum number of memory accesses
- An example is run-length encoding, which is a very simple compression technique that replaces consecutive occurrences of a given symbol with one occurrence plus a count of how many times the symbol occurs

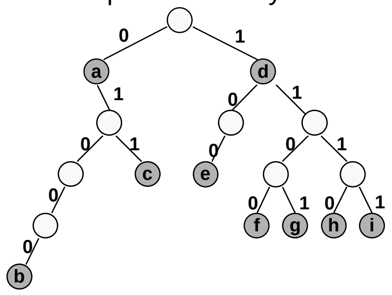
Search on prefix length using multibit tries

- Search in a binary trie can be rather slow, because one bit at a time is inspected => in the worst case, 32 memory accesses are needed for an IPv4 address
- Search operation can be speedup by inspecting several bits at a time - for example, inspecting four bits at a time would lead to only 8 memory accesses in the worst case for an IPv4 address
- Number of bits to be inspected is called stride and can be constant or variable
- A trie allowing inspection of bits in stride of several bits is called a multibit trie => multibit trie has **2^k** children, where **k** is the stride => arbitrary prefix lengths not supported, because tries are traversed in strides of several bits

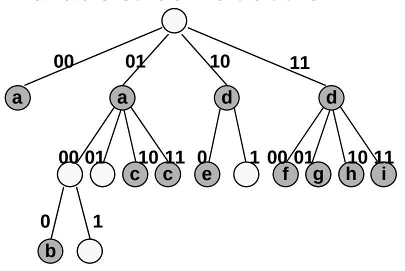
Multibit trie example 1

- Prefixes **a** and **d** are expanded to length 2 and prefix **c** has been expanded to length 4 (rest of the prefixes remain unchanged)
- Height of the trie has been decreased and so has the number of memory accesses when doing a search

Uncompressed binary trie



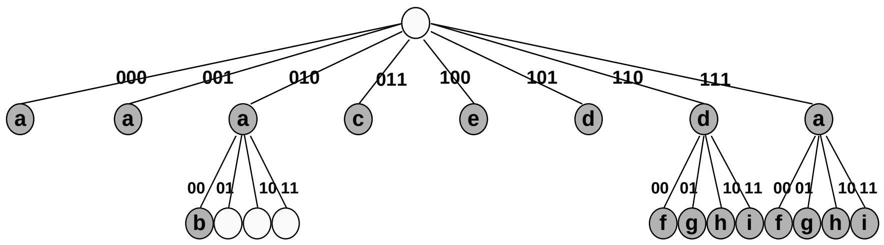
Variable stride multibit trie



Multibit trie example 2

- An alternative multibit trie of the example case
  - prefixes **a** and **d** have been expanded to length 3
  - rest of the prefixes remain the same as before expansion
- When an expanded prefix collides with an existing one, forwarding information of the existing one must be maintained (to respect the longest match)

Fixed stride multibit trie



Search in a multibit trie

- Search in a multibit trie is essentially the same as search in a binary (1-bit) trie - successively look for longer prefixes that match and the last one found is the longest match prefix for a given address
- Multibit tries do linear search on length as do binary tries, but the search is faster because the trie is traversed using larger strides
- A multibit trie is a fixed stride system, if all nodes at the same level have the same stride size, otherwise it a variable stride system
- Fixed strides are simpler to implement than variable strides, but usually consume more memory

Choice of stride size and update of tries

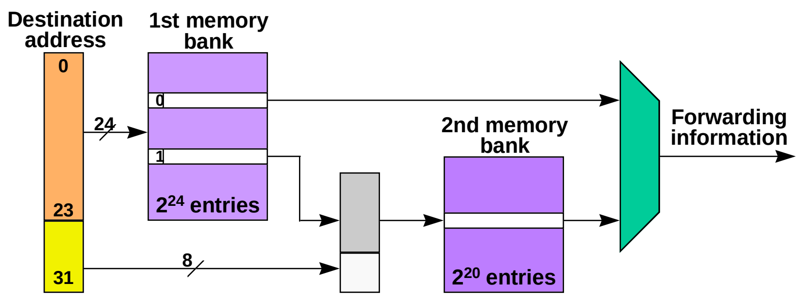
- Choose of stride size is a trade-off between search speed and memory consumption
  - in the extreme case, a trie with a single level would be made (stride size = 32) and search would take only one memory access, but a huge amount of memory would be required 2^{32} entries
  - a natural way to choose stride size and memory consumption is to let the binary trie structure determine this choose
- Update bounds determined by stride size
  - a multibit trie with several levels allows, by varying stride k, an interesting trade-off between search time, memory consumption and update time - larger strides make faster search
    => memory consumption increases and updates will require more entries to be modifies (because of expansion)

Multibit tries in hardware

- In core network routers, lookup times are very short and lookup algorithms are implemented in hardware to obtain required speed
- Basic scheme uses two level multibit trie with fixed strides
  - 24 bits at the first level and 8 at the second level
- In backbone routers, most of the entries have a prefix length of 24 bits or less => longest match prefix found in one memory access in the majority of cases
- Only a small number of sub-entries at the 2nd level
- To save memory, internal nodes not allowed to store prefixes
  => prefixes corresponding to internal nodes expanded to 2nd level
  => result is a multibit trie with disjoint expanded prefixes - 1st level has 2^{24} nodes and is implemented as a table with the same number of entries

Multibit tries in hardware (cont.)

- An entry at the 1st level contains either the forwarding information or a pointer to the corresponding sub-trie at the 2nd level - two bytes needed to store a pointer/forwarding information
  => a memory bank of 32 Mbytes is needed to store 2^{24} entries



Multibit tries in hardware (cont.)

- Number of sub-tries at the 2nd level depends on the number of prefixes longer than 24 bits
- 2nd level stride is 8 bits => a sub-trie at the 2nd level has 2^8=256 leaves
- Size of 2nd memory bank depends on the expected worst case prefix length distribution, e.g. 2^{20} one byte entries (a memory bank of 1 Mbytes) supports a maximum of 2^{12}= 4096 sub-tries at the 2nd level
- Lookup requires a maximum of two memory accesses
  - memory accesses can be pipilined or parallelized to speed up performance
- Since the first stride is 24 bits and leaf pushing is used, updates may take a long time in some cases

Prefix range search

- One way to get rid of length dimension is to transform prefixes to a unique length - full expansion yields 32-bit long prefixes (IPv4)
- A prefix determines a well-defined range of addresses and search can be accelerated by using range end-points instead of every single address
- Multiway search can be applied to reduce search steps
- In a multiway search tree, internal nodes have k branches and k-1 keys - keys are extra “prefixes” that are used to direct search to right direction when entering a node in which no match is found
- If an entire node of a search tree can be designed to fit into a single cache line, multiway search becomes especially attractive

Complexity comparison of some lookup schemes

Scheme	Worst case lookup	Update	Memory
Binary trie	O(W)	O(W)	O(NW)
Path-compressed trie	O(W)	O(W)	O(N)
K-stride multibit trie	O(W/K)	$O(W/K+2^K)$	$O(2^K NW/K)$
Binary search on prefix length	$O(\log_2 W)$	$O(N\log_2 W)$	$O(\log_2 W)$
Binary range search	$O(\log_2 N)$	O(N)	O(N)

W - length of address in bits, N - number of prefixes in a prefix set

Source: IEEE Network, March/April 2001

Lookup scalability and IPv6

- On scalability point of view, important aspects are the number of entries in a lookup table and the prefix length
- Multibit tries improve lookup speed with respect to binary tries, but only by a constant factor on the length dimension
  => multibit tries scale badly to longer addresses (128 bit in IPv6)
- Binary search on length has logarithmic complexity with respect to prefix length => scalability very good for IPv6
- Range search has logarithmic lookup complexity with respect to the number of entries, but independent of prefix length
  => if the number of entries does not grow excessively, range search is scalable for IPv6