

dress is *self-identifying* (i.e., the size of the prefix can be computed from the address itself, a hash table works well for IP lookup. In a classless world, however, hashing does not work well. Thus, an alternative must be used.

9.20.1 Searching By Mask Length

The key to understanding CIDR lies in observing that the address block an ISP assigns to a subscriber always has a longer address mask than the block owned by the ISP. Thus, the goal of route lookup is *longest prefix match (LPM)*. That is, given a destination address, D , find the entry in the routing table that has the longest prefix of bits that match the bits of D .

The simplest LPM algorithm iterates over all possible divisions between prefix and suffix. That is, given a destination address, D , the algorithm first tries matching all 32 bits of D , then 31 bits, and so on. For each possible size, M , the router extracts M bits from D , assumes the extracted bits form a network prefix, and looks up the prefix in the table. The algorithm stops once a match has been found.

The disadvantage of trying all possible lengths should be obvious: doing so is extremely slow because the algorithm performs up to 32 lookups for each datagram. The worst case occurs when no route exists, but even when it finds a route, the iterative approach searches the table many times. Except in cases where a routing table contains host-specific routes, the algorithm wastes time checking host bits. More important, the algorithm performs 31 unnecessary lookups before it can decide to follow a default route (in many routing tables, the default route is heavily used).

9.20.2 Binary Trie Structures

To avoid inefficient searches, a hierarchical data structure is used for classless lookup. The most popular data structures are variants of a *binary trie* in which the value of successive bits in the address determine a path from the root downward.

A binary trie is a tree with paths determined by the data stored. To visualize a binary trie, imagine that a set of 32-bit addresses is written as binary strings and redundant suffixes are removed. What remains is a set of prefixes that uniquely identify each item. For example, Figure 9.12 shows a set of seven addresses written in binary and the corresponding unique prefixes.

As Figure 9.12 illustrates, the number of bits required to identify an address depends on the values in the set. For example, the first address in the figure can be uniquely identified by two bits because no other addresses begin with 00 . However, five bits are required to identify the last item in the table because the 4-bit prefix 1011 is shared by more than one item.