First mini-project: Forwarding traffic in the Internet

João Luís Sobrinho

I. Introduction

Every interface connected to the Internet has an IP address which is a string of 32 bits. Every data-packet sent into the Internet carries, in its header, the address of the interface to which the data-packet is destined. Every router on the Internet has a forwarding table that tells it how to forward data-packets one-hop closer to the destination. The forwarding table of a router is stored in expensive, high-speed memory. It cannot contain entries for all possible IP addresses, of which there are 4G. Rather, the forwarding table entries of a router map *prefixes* to *next-hops*. A prefix is a string of less then 32 bits that is advertised Internet-wide by a routing protocol. A prefix represents all addresses whose first bits coincide with the prefix. For example, prefix 01 represents all addresses whose first two bits are 01. A next-hop is an identifier that singles out a unique neighbor of the router. Figure 1 (left) shows a forwarding table, where $*$ is the empty prefix that represents all addresses.

A prefix q is more specific than a prefix p if the addresses represented by q are contained in the addresses represented by p. In other words, prefix q is more specific than prefix p if string q starts string p. For example, 110 is more specific than 11, since all addresses that begin with 110 also begin with 11. The empty prefix is the less specific of all prefixes. There can exist entries for prefixes at different levels of specificity in a forwarding table. The *longest prefix match rule* specifies that an address is matched to the longest of the prefixes that contains the address. For example, in the forwarding table of Figure 1, a data-packet with destination address starting with 11010100 will match prefix 110 and will be forwarded to next-hop 4; a data-packet with destination address starting with 00011101 will match prefix 00 and will be forwarded to next-hop 2; and a data-packet with destination address starting with 01101011 will match the empty prefix and will be forwarded to next-hop 1.

Forwarding table

prefix	next-hop
*	1
00	2
10	2
11	3
010	3
110	4

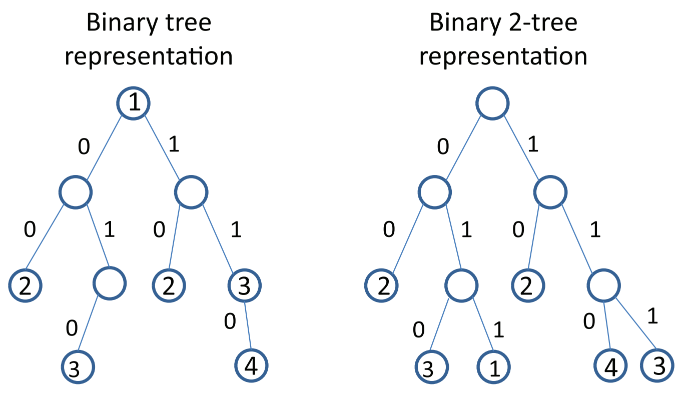

Fig. 1. A forwarding table, its binary tree representation, and its binary 2-tree representation.

II. Representation of a forwarding table

A forwarding table is accessed to look up an address at the rate at which data-packets arrive, which rate may be very large. A forwarding table also needs to be updated constantly to account for the announcement of new prefixes and the withdrawal of old prefixes, typically occurring at a much slower rate than that of the arrival of data-packets. It is of paramount importance to store a forwarding table efficiently and to have efficient algorithms for address look-up and forwarding-table maintenance.

A forwarding table is represented as a binary tree where each node corresponds to a binary string. Every prefix of the forwarding table has a node in the binary tree that indicates the next-hop for the prefix. The node corresponding to a prefix is reached from the root by analyzing each successive bit of the prefix, going to a left-child node if it is 0, and to a right-child node if it is 1. All leaves of the binary tree are prefixes. Internal nodes do not have to be prefixes. Such a binary tree is called a *binary prefix tree*. Figure 1 shows a forwarding table (left) and its binary prefix tree (center).

Introducing a new prefix in a binary prefix tree is straightforward. Just follow the bits of the new prefix in succession, adding new binary nodes to the tree as needed. Deleting a prefix from the binary tree is less straightforward. It may require deleting internal nodes that no longer lead to any prefix. For example, deletion of prefix 010 from the binary prefix tree of Figure 1 implies that both nodes 010 and 01 are deleted.

The binary prefix tree representation of a forwarding table does not lead to a very efficient address look-up algorithm. In an address look-up, one would probably like to use each of the bits of the address in turn to steer down on the binary prefix tree and retrieve the next-hop when it is not possible to proceed any further. Suppose that the binary prefix tree is the one in Figure 1 and we are looking for the next-hop of an address starting with 011. The first bit 0 would guide us left on the binary prefix tree, the second bit 1 would guide us right, but the third bit 1 would guide us nowhere. However, the node corresponding to the string 01 does not have a next-hop associated with it to be retrieved. A better way to represent a forwarding table for the purpose of address look-up is to create a *binary prefix 2-tree* where each node has either zero or two children and all next-hops are concentrated in the leaf nodes. The binary prefix tree is enlarged by creating new leaf nodes whose next-hops are inherited from the nearest ancestor that has a next-hop. The interior nodes no longer require any next-hop. Figure 1 shows a binary prefix tree (center) and the corresponding binary prefix 2-tree (right). An address look-up in a binary 2-prefix tree is straightforward. Follow each bit of the address in turn until you hit a leaf node and retrieve the next-hop from there. Suppose that the binary prefix 2-tree is the one in Figure 1 (right) and you are looking for the next-hop of an address starting with 011. The first bit 0 guides you left, the next bit 1 guides you right, and the third bit 1 guides you right, from where you retrieve next-hop 1.

III. Your assignment

What you have to do.

- Implement the following functions: (i) *ReadTable* that reads from a file a forwarding table matching prefixes to next-hops into a binary prefix tree representation; (ii) *AddPrefix* which adds a prefix to the binary prefix tree; (iii) *DeletePrefix* which deletes a prefix from the binary prefix tree; (iv) *PrintTable* that prints a binary prefix tree into the standard forwarding-table format matching prefixes to next-hops. For simplicity, you may assume that addresses have 8 bits, rather than 32. The $*$ represents the empty prefix. (50% of the grade.)

- Implement the following functions: (i) TwoTree that converts a binary prefix tree into a binary prefix 2-tree; (ii) AddressLookUp that receives an address and returns the corresponding next-hop. (50% of the grade.)

What do you have to deliver, how, and when.

- You have to deliver your code and a report of no more than three pages containing a text explanation of your algorithms, their pseudo-codes, and a short discussion.
- The code and the report should be sent in a .zip file to my email address with subject p1.<group number>.zip where <group number> is your group number. You will also have to deliver a printed version of the report.
- The deadline for the .zip file is October 16, 2014, 23:59. The printed report should be delivered at the class on October 19.

How I will evaluate your assignment.

- Write your report and your pseudo-codes clearly, and present a commented code. Your writing is your way of communicating your ideas to others so that your work can be well understood and publicized. In addition, writing clearly forces you to think clearly about what you are doing.
- Be sure to test your code for correctness. I will also take into consideration the efficiency of your algorithms, but mostly in asymptotic terms.
- I will have a discussion with you about your report and will test your code at the end of the semestre, jointly with the other assignments, but I may give some feedback before the second assignment.