Packet Switching practice: Route Lookup

1. Goals

- Introduce the students to the next hop search problem (route lookup) in a CIDR router, as well as with its complexity, in a practical scenario.
- Introduce the students to the basic mechanisms of route lookup.
- Learn state-of-the-art algorithms.
- Implement a route lookup algorithm that improves the performance of the linear search algorithm, whose implementation is used as reference (baseline).
- Evaluate the performance of the implemented route lookup algorithm, compared to the linear search algorithm.
- Present the challenge of improving existing algorithms.

2. Algorithm selection and implementation

Review the proposed algorithms below and choose one that you consider could outperform the reference one (e.g., in terms of search speed, memory consumption, etc.). This reference algorithm is also described in [1] (page 8, Linear Search of Hash Tables).

It is necessary to choose and implement one of these algorithms.

a) **Compressed binary trie *(Patricia trie)*** [1][2]. **(1.5 pts, 10/10)**. This algorithm is explained in the theory class.
b) **Binary search with hash** [1]. **(2 pts, 13.3/10)**. To understand this method, please read:
   - Section 1: introduction. Pages 2 and 3.
   - Section 2: existing algorithms. Page 4.
   - Section 3: linear and binary search. Pages 8 to 12.
   - Section 5.1: creating a routing database. Pages 22 and 23.
     This algorithm works better than the linear case for most real routing tables, but not always. You must understand when and why.
c) **LC-tries** [3]. **(2 pts, 13.3/10 pts.)**. Which is the algorithm implemented in Linux [2], is an optimization of the Patricia trie, so you will need to implement option a) before.

3. Specifications

The program implemented must meet the following requirements:

- Programming language: C.
- Operating System: Linux.
- Memory Limit: 50 MB of RAM memory. Consider a maximum number of 25.000 prefixes of any length. Suppose that the number of possible next hops or interfaces is lower than 32000.
- A program in C must be developed. The executable file must be `my_route_lookup`, and it will take the following two parameters:

  `./my_route_lookup` *FIB InputPacketFile*

  where:

- *FIB*: name of the ASCII file containing the FIB (Forwarding Information Base).
- *InputPacketFile*: name of the ASCII file containing a list of destination IP addresses to be processed, separated by new line characters (`\n` in C language).

- The FIB file represents a simplified Forwarding Table[1], each line consisting of a <CIDR_Network_Prefix, Output Interface> tuple. Both fields will be separated by a single tab character (`\t` in C language). For example:

Network Prefix	*Output Interface*
192.163.10.0/24	1
192.163.0.0/16	2
193.110.27.192/2	3

- For each destination IP address included in the Input File named *InputPacketFile*, the program will obtain the appropriate output interface to forward the packet according to the *FIB*.
- Assume the input files will never have any wrong addresses or prefixes.
- The router interfaces are numbered from 1 on. **A 0 represents the non-existence of an interface.**
- If there is no default gateway, the packet is discarded.
- You must choose how to model the default gateway in your implementation.
- As an output, the program will generate a single file, *OutPutFile*, where a *<IPaddress, OutIfc, AccessedNodes, ComputationTime>* tuple will be stored in each line, for each IP address in the input file (*InputPacketFile*). The fields of such tuple are defined as follows:

  - *IPaddress*: an IP address read from the *InputPacketFile*.
  - *OutIfc*: output interface through which the packet should be routed ("`MISS`" should be printed instead if no outgoing interface is found).
  - *AccessedNodes*: number of nodes visited (memory accesses) required to obtain the appropriate outgoing interface for a given IP address. **Do not consider the root node for node counting**.
  - *ComputationTime*: required time to get the appropriate outgoing interface for the destination IP address. This time ONLY includes the outgoing interface calculation time in the FIB, but it DOES NOT INCLUDE the time required for reading/writing files to disk. The calculation time will be an integer value in nanoseconds.

- The *OutPutFile*, will follow this format:
  - The name of this file is formed by the input file name plus "`.out`".
  - It will have a tuple per line.
  - The fields of each tuple will be separated by "`;`". After the last field (Calculation time) a single new line character will be printed.
  - No blank lines will be printed between two tuples.
  - A single blank line will be printed after the last tuple.
  - The total number of processed destination IP addresses will be printed, next, the average number of accessed nodes by lookup and finally the average calculation time for the processed addresses. The last two fields will be printed with two decimals.

[1] Please remember that a real complete FIB would include, apart from the outgoing interface, the IP address of the next hop as well as link layer forwarding information associated to that neighbour (typically an Ethernet MAC address). It would also include the encapsulation information for the packet, the handler or hardware/software function to be invoked with those parameters and the pointer to the IP packet processed.

Likewise, the consumed memory and CPU time will be also printed. The format will be as follows:

```
Number of nodes in the tree = NNNNN
Packets processed= X
Average node accesses= YY.YY
Average packet processing time (nsecs)= Z.ZZ
Memory (Kbytes) = KB
CPU Time (secs)= T.TTTTTT
```

These parameters related to the performance of the algorithm along with the number of correct lookups will be considered when grading the lab assignment.

4. Example

A complete example for the required program, including input and output files, is shown next:

File: r.txt	
192.163.10.0/24	1
192.163.0.0/16	2
193.110.27.192/26	3

File: i.txt
192.163.10.123
195.50.230.11
193.110.27.224

Command line	File: i.txt.out
pract> ./my_route_lookup r.txt i.txt pract>_	192.163.10.123;1;2;500 195.50.230.11;MISS;1;200 193.110.27.224;3;1;200 Number of nodes in the tree = 4 Packets processed = 3 Average node accesses = 1.33 Average packet processing time (nsecs) = 300 Memory (Kbytes) = 21258 CPU Time (secs) = 0.044427

5. Provided files

To make it easy for the student to reproduce the output file format, a set of C files with functions is given. These functions will be useful to develop the program to be delivered. These files are:

- `io.c/h:` These files include the necessary code to parse a routing table, read a file with IP addresses to be looked-up and generate a file with the format specified in the sections above.
- `utils.c/h:` These files contain a hash function and another function that calculates an IP mask from a given prefix length. The hash function might be of use or not depending on the implemented algorithm. The students may use other hash tables if they consider it convenient.
- `routing_table.txt/routing_table_simple.txt:` Routing tables that you must process to ensure that the developed program works properly.
- `prueba0/1/2/3.txt:` Files with IP addresses that must be routed as specified in this document.
- `Makefile:` A makefile example that shows how to generate the required executable file.

You must change this file by modifying the list of files scanned by the makefile to match those of your solution.

- `linearSearch`: Executable file that makes a linear search and that will be used as a reference to compare with the solution developed by the students.

Suggestions:

1. As a first step, implement a binary tree module with a set of functions for creating, releasing, printing, etc. the content of nodes and binary trees (printing will be useful for visualizing trees while debugging).
2. Start testing with the simplest route tables and gradually increase the complexity of the cases you want to test.
3. For simplicity, create the uncompressed trie first with all the routes and then compress it, freeing the memory of the nodes that are removed (this will only reduce the effective memory used in some cases because the OS allocates larger memory blocks to processes than what the user requests, and the block may contain nodes that have not been released).
4. Ensure that tables with the default route 0.0.0.0/0 work correctly.

6. Lab. Assignment Delivery

The deadline for submitting the lab assignment is Wednesday, April 11 2025, 23:59 CET. The student must deliver a .zip file through the *Aula Global* delivery system, meeting the following requirements:

1. The zip file must be called GRUPOXX (even if you are in the English group). So if your group number is 5, the assignment must be GRUPO05.zip.
2. You must include all the code files (both yours and provided to you as part of the lab assignment) needed to compile and generate the executable.
3. You must include a Makefile file to be run with the linux tool make. The Makefile contains a configuration that supports: “make all”, “make my_route_lookup” and “make clean” as follows (adapt it to your needs):

```
SRC = my_route_lookup.c io.c io.h utils.c utils.h
#CFLAGS = -Wall -O3
CFLAGS = -g

all: my_route_lookup

my_route_lookup: $(SRC)
gcc $(CFLAGS) $(SRC) -o my_route_lookup -lm

.PHONY: clean   # directive indicating that “clean” is not a file

clean:
rm -f my_route_lookup
```

In other words, the target "all" depends on the target "my_route_lookup", and the target my_route_lookup" is generated as long as: the file "my_route_lookup" does not exist or any of the files in $(SRC) changes - where $(SRC) is the list with all the source files of your program (both yours and provided to you as part of the lab assignment).

4. You must include a file named "`nias.txt`": This file must have two lines with the NIA (student ID) of the two students that developed the solution. For example:

```
100012345
100023456
```

5. Important: Please do not zip a folder, but the collection of files (code files + auxiliary files such as the ASCII report, nias.txt, etc).

- Every submission must meet these requirements. Otherwise, it will not be graded.

The number associated to each group will be the one selected in the form at Aula Global. The form will be open until the end of the second session of the lab.

It will be considered that those students who do not perform such registration before the deadline will not do the lab assignment. Only in the case that the number of students of a given degree were odd, the student without group would be regrouped into a 3-student group.

IMPORTANT NOTE

It is the responsibility of the students to protect their own code. With the delivery of this lab assignment, the student implicitly acknowledges the authorship of the delivered code. The only non self-developed code to be used is the one given to the students by the teachers. AI tools can only be used to help debugging.

The use of Open Source libraries for hash tables should be approved by the teachers. An automated detection system will be employed to detect plagiarism. Source code plagiarism will be reported to the academic authorities and both plagiarising and plagiarised students will be punished.

7. Evaluation conditions

No algorithm will be accepted if:

- The program does not compile.
- The program leaks memory.
- Run time exceptions are detected on the evaluation tests.
- More than 50 MB of RAM are used for 25000 routes with prefix of different lengths from 0 to 32 bits. Please notice that these conditions are more strict than in real cases.
- The next hop is not calculated correctly in all cases.

The next hop information will be simply an unsigned integer (in reality, this would be an index to an array of next hops of the router, including an IP address, output interface number, etc that we don't consider in this practice for the sake of simplicity). You can assume that the number of possible next hops is lower than 32000.

The assessment has two parts:

1. **Code submission.** The performance improvement over the reference algorithm, as well as the clarity and organization of the code will be evaluated.
2. **Class assessment.** The understanding of both the problem and technical characteristics of the delivered solution will be evaluated in two parts.

a. In the second session of the lab, the students will be asked to explain and justify, technically speaking, their solution as well as describe how they are going to implement it.
b. A face-to-face exam after the code submission where the students will be asked individually about specific details of the design and the implementation of the submitted code. The students can be asked about technical details of the algorithm concerning its space and time complexity as well as its drawbacks or limitations. **This exam will take place in parallel to the second session of the MPLS lab (23-24 of April).**

8. References

[1] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable High Speed Prefix Matching. ACM Transactions on Computer Systems, 2001.
[2] Route lookup en Linux - https://vincent.bernat.ch/en/blog/2017-ipv4-route-lookup-linux
[3] "IP-address lookup using LC-tries," IEEE Journal on Selected Areas in Communications, 17(6):1083-1092, June 1999. https://dl.acm.org/doi/10.1109/49.772439