UNIVERSITÀ DI PISA

Distributed Systems and Middleware Technologies

Distributed File System with Pastry

Iacopo Canetta

Francesco Berti

ANNO ACCADEMICO 2024/2025

Contents

1 Introduction

This project consists in the implementation of a peer-to-peer distributed file storage using a distributed hash table for locating the files among the nodes. The distributed hash table is based on Pastry protocol.

2 Pastry

Pastry implements a logical ring of nodes, and each node has an ID. The files are hashed to get a key, and the key is used to identify which node is responsible for that file, checking which one has the longest common prefix. When a file is accessed or stored, a request is sent to the current node. When a node receives a request, it uses a routing table, structured as a tree, for finding the next hop for the request. Every time a row of the routing table is accessed, b bits of the prefix of the key are used to identify the correct column. If the b bits of the prefix are owned by the current node, the next row is visited, otherwise the address of the next hop is retrieved.

Whenever an address is not found or the whole table is crossed, an extra table is checked, called leaf set and containing the nodes with id close to the current node; if no node in the leaf set has a longer shared prefix with the key than the current node, then the current node is responsible for the file. This routing table, using N nodes, guarantees a maximum number of hops equal to logn.

3 Implementation

The implementation of the system is divided in two parts: one for the distributed file system, created in Erlang, and a webserver realized in Java, also thanks to the Frameworks and implementions of the Java Servlets.

3.1 Erlang

The Erlang code is divided in multiple modules, in order to isolate functions and keep them short enough.

3.1.1 Util.erl, File_handler.erl, Network.erl

This files are created for having simple utility functions. *File_handler.erl* contains the utility functions for dealing with files, *Network.erl* contains simple functions for starting the Erlang nodes and settign the cookies, in order to receive messages from different machines, and *Util.erl* contains generic utility functions.

3.1.2 Key_gen.erl

Key_gen.erl is the file that implements the key system for Pastry. The keys are generated hashing strings and represented as *bitstring*s; this type is chosen because it can be used for representing streams of any number of bits, differently from the *binary* type, which is exclusively for multiples of 8 bits.

*Bitstring*s, however, have little documentation and almost no function dedicated, so *Key_gen.erl* also implements the functions for comparing these numbers and handling them as hexadecimals, since Pastry requires groups of 4 bits for the routing.

3.1.3 Routing.erl

Routing.erl represents the core of Pastry protocol, implementing the Routing Table. The routing table is represented as a Tuple containing the TableKey, so the key of the node owning the table, and the table. The table is a list of 32 lists, representing the rows; the single entries of the rows are represented by a tuple containing the hexadecimal identifying the column and the node information.

During the routing or any table modification, instead of doing the whole routing, this approach requires of detecting the number of hexadecimals in common in the prefix, which represents the index of the row, and the first hexadecimal not in common, which represent the Column key.

This module contains also the functions for modifying the table and some utility functions relative to it.

3.1.4 Leaf_set.erl

Leaf_set.erl implements the Leaf Set; the Leaf Set is represented as a Tuple of two lists, *Left* and *Right*, representing the two halves of it. The two list contain at most $L2$ elements, represented as tuples key-node information.

The module also offerts functions for modifying the leaf set and some utility relative to it.

3.1.5 Node_actions.erl

Node_actions.erl implements some utility functions used by the upper modules to interact with files, sending messages, or broadcasting messages to lists, routing tables, or leaf sets.

The most important function is *full_route()*, which implements the full navigation of the routing table, and, in case of no match or it reaches the bottom, the leaf set is analyzed; the function returns the information of the routed node, or *route_end* if the current node is responsible for the key in input.

3.1.6 Pastry_actions.erl

Pastry_actions.erl implements all the functions for handling the pastry requests and responses to the other nodes in the network, plus the handling of the self messages for peridoc events. The main functions are:

- *join()*: function used for responding to the *join* message with the routing table row, and for forwarding the message to the next node of the routing; function used by the node to join the network through another known node;
- *join_res_handle()*: function used for updating the routing table and leaf set after a *join_response*;
- *exit()*: function for handling the exit of another node from the network;
- *keepalive()*: sends keepalive to all known nodes;
- *update_keepalive()*: updates the keepalive of given nodes;
- *keepalive_res()*: responses to the keepalive to make sure that, even if a node knows another but not viceversa, the one knowing still receives keepalives from the other;
- *check_expired_nodes()*: checks which nodes are disconnected and updates the tables;
- *share_info()*: shares the info about the nodes with the known nodes.

3.1.7 Backup_actions.erl

Backup_actions.erl implements all the functions for sending and handling the backups. For design choice, the backups are propagated to the whole leaf set and updated according to it. The main functions are:

- *backup()*: function used for propagating a file to the leaf set for backup;
- *backup_res()*: function used for saving a backup sent by other node;
- *backup_remove()*: function used to communicate the leaf set to remove a file from backup;
- *new_leaf_backup()*: function used to check if a node is added to the new leaf set, and in that case it receives a backup of each file;
- *old_leaf_backup()*: function used to check if a node is removed from the new leaf set, and in that case it receives a message to delete the backup folder;

- *update_leaf_backup()*: function used for checking in a list of nodes which ones entered the leaf set and which ones exited, in order to propagate or remove backups;
- *backup_update()*: function used to update the owner of the backups of a failed node. The routing is performed and, if the current node is the owner, the node moves the files to its folder, otherwise it forwards the request;
- *backup_find()*: request to find the owner of a file. If reaches the end, a *backup_found* response is sent;
- *backup_found()*: updates the owner of a file on a *backup_found*.

3.1.8 Web_responses.erl

Web_responses.erl implements all the functions for handling and responding to the webserver. The main functions are:

- *find_store()*: function that, given a filename, finds a storage for it using pastry routing and responds to the web server;
- *store()*: function that, given a filename, it stores it on the node;
- *find()*: function that finds a file and sends it back to the webserver;
- *delete()*: function that, given a filename, finds for it using pastry routing, deletes it, and sends and ack to the web server;
- *get_file_res()*: function that starts a flooding for retrieving all the names of the files in the network; it uses a blacklist on the *Msg_Id* for avoiding infinite flooding;
- *get_files_res_handle()*: function that updates the list of files with the new nodes received during the flooding;
- *all_files_res()*: function that responds to the webserver with the list of files;
- *check_expired_blacklist()*: function called periodically for cleaning the expired entries of the blacklist.

4 Testing and Results

Once Mininet and ONOS are initialized, the network is tested to guarantee that it works as inteneded by the VPLS. The network is tested both with the *ping* and both with a configured broadcast ethernet frame. The broadcast is executed using the python script *frameBroadcaster.py*.

For checking that the VPLS are working as intended, we used *Wireshark*, checking which interfaces and devices receive the ethernet frames broadcasted. Different topologies were initialized as *net.json* and run. The topologies and some of the screenshots of wireshark are shown below, to prove how only the interfaces in the same VPLS are reached by the broadcast.

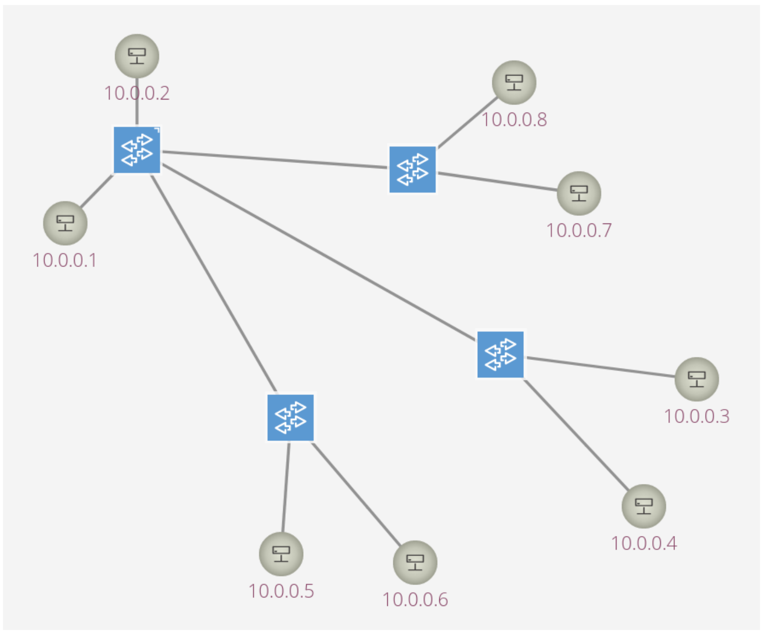
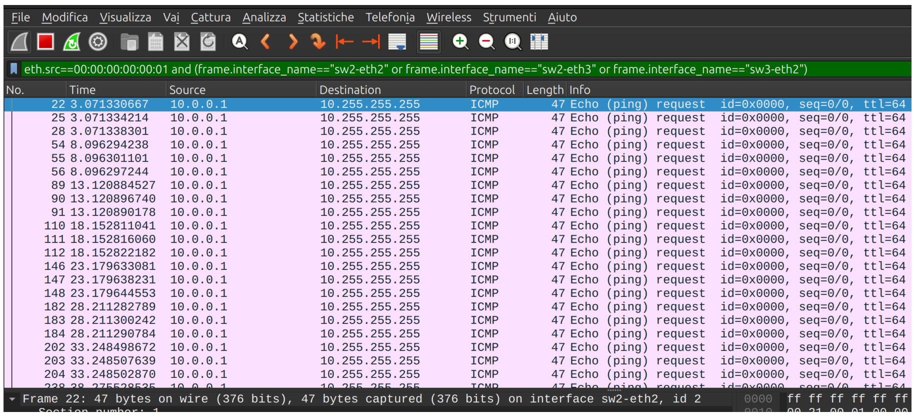


Figure 1: Topology 1

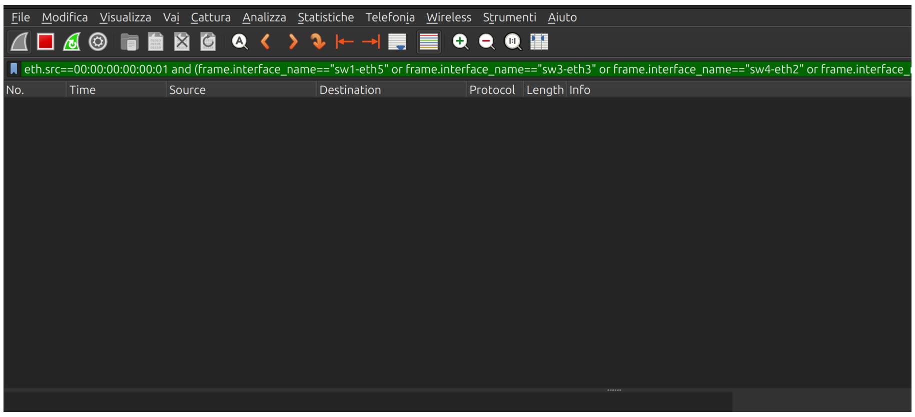

File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

```
eth.src==00:00:00:00:00:01 and (frame.interface_name=="sw2-eth2" or frame.interface_name=="sw2-eth3" or frame.interface_name=="sw3-eth2")

No.     Time         Source       Destination       Protocol  Length Info
     22 3.071330667  10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     25 3.071334214  10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     28 3.071338301  10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     54 8.096294238  10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     55 8.096301101  10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     56 8.096297244  10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     89 13.120884527 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     90 13.120896740 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     91 13.120890178 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    110 18.152811041 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    111 18.152816060 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    112 18.152822182 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    146 23.179633081 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    147 23.179638231 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    148 23.179644553 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    182 28.211282789 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    183 28.211300242 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    184 28.211290784 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    202 33.248498672 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    203 33.248507639 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    204 33.248502870 10.0.0.1     10.255.255.255    ICMP         47 Echo (ping) request  id=0x0000, seq=0/0, ttl=64

▾ Frame 22: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface sw2-eth2, id 2    0000  ff ff ff ff ff ff
```

Figure 2: Broadcast example 1: servers reached



```
eth.src==00:00:00:00:00:01 and (frame.interface_name=="sw1-eth5" or frame.interface_name=="sw3-eth3" or frame.interface_name=="sw4-eth2" or frame.interface_
```

```
No.   Time   Source   Destination   Protocol   Length   Info
```

Figure 3: Broadcast example 1: all other interfaces not reached

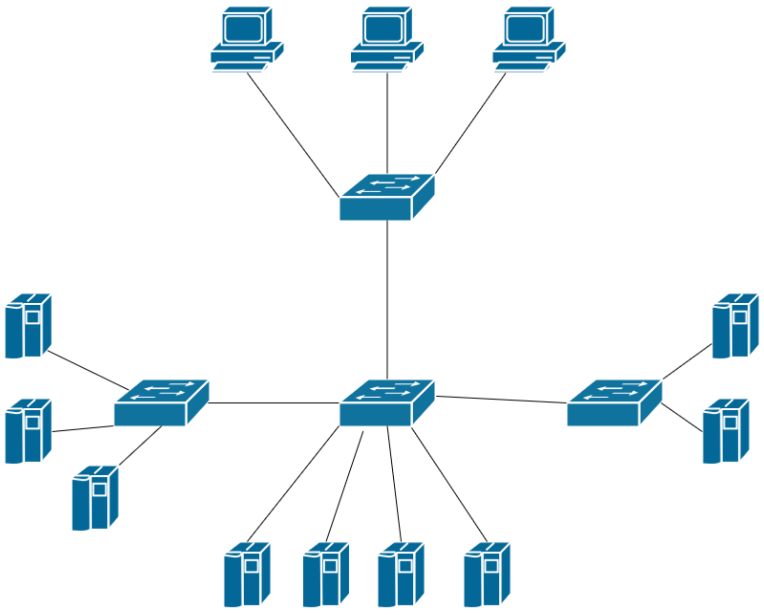
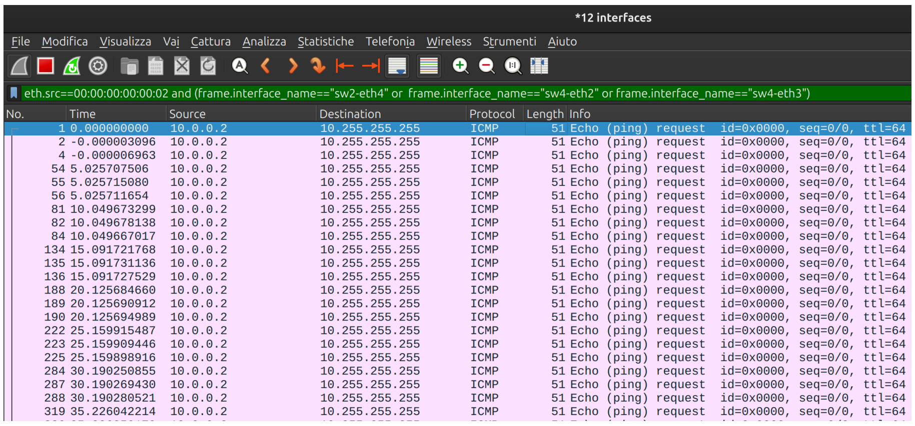
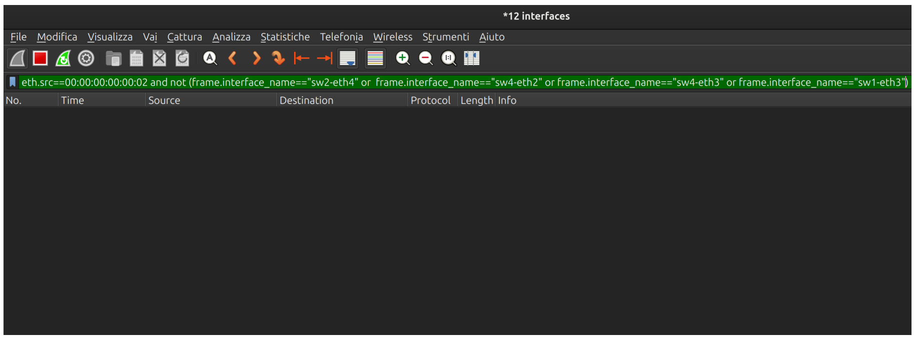


Figure 4: Topology 2


*12 interfaces
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

```
eth.src==00:00:00:00:00:02 and (frame.interface_name=="sw2-eth4" or  frame.interface_name=="sw4-eth2" or frame.interface_name=="sw4-eth3")
No.   Time          Source     Destination      Protocol Length Info
    1 0.000000000   10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    2 -0.000003096  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    4 -0.000006963  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
   54 5.025707506   10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
   55 5.025715080   10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
   56 5.025711654   10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
   81 10.049673299  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
   82 10.049678138  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
   84 10.049667017  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  134 15.091721768  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  135 15.091731136  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  136 15.091727529  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  188 20.125684660  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  189 20.125690912  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  190 20.125694989  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  222 25.159915487  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  223 25.159909446  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  225 25.159898916  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  284 30.190250855  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  287 30.190269430  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  288 30.190280521  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
  319 35.226042214  10.0.0.2   10.255.255.255   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
```

Figure 5: Broadcast example 2: servers reached


*12 interfaces
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

```
eth.src==00:00:00:00:00:02 and not (frame.interface_name=="sw2-eth4" or  frame.interface_name=="sw4-eth2" or frame.interface_name=="sw4-eth3" or frame.interface_name=="sw1-eth3")
No.   Time          Source     Destination      Protocol Length Info
```

Figure 6: Broadcast example 2: all other interfaces not reached

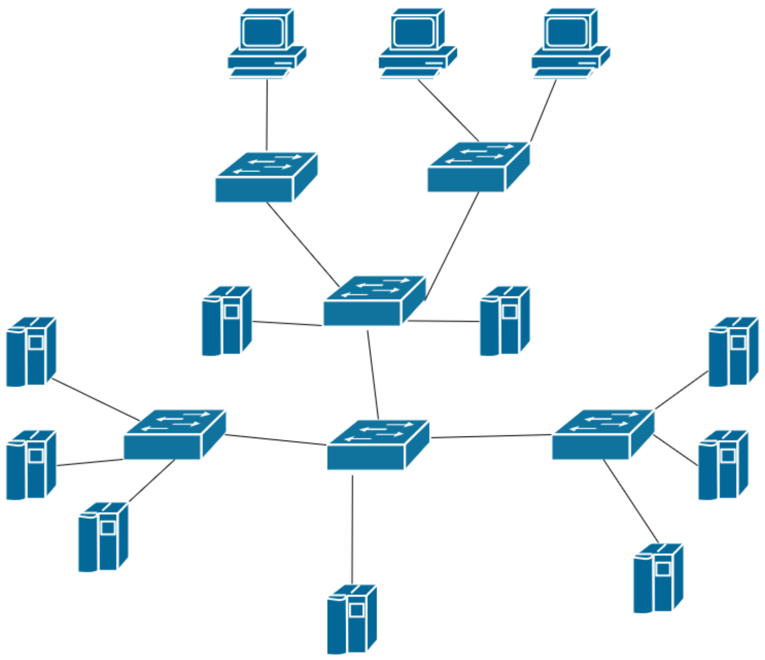
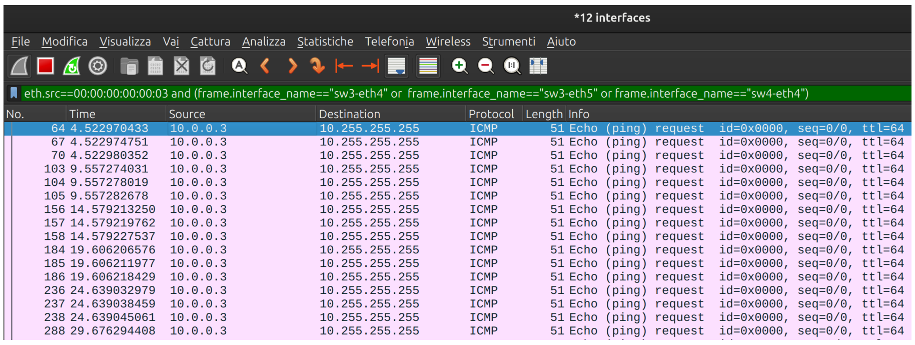
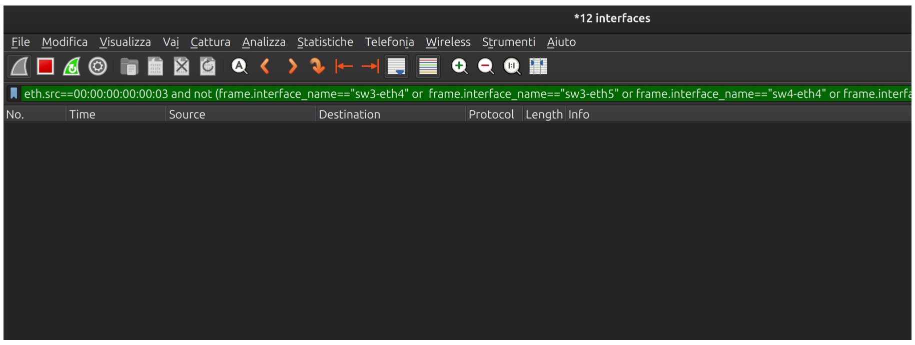


Figure 7: Topology 3


*12 interfaces
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

```
eth.src==00:00:00:00:00:03 and (frame.interface_name=="sw3-eth4" or  frame.interface_name=="sw3-eth5" or frame.interface_name=="sw4-eth4")
No.      Time          Source          Destination       Protocol  Length Info
      64 4.522970433   10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
      67 4.522974751   10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
      70 4.522980352   10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     103 9.557274031   10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     104 9.557278019   10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     105 9.557282678   10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     156 14.579213250  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     157 14.579219762  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     158 14.579227537  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     184 19.606206576  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     185 19.606211977  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     186 19.606218429  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     236 24.639032979  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     237 24.639038459  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     238 24.639045061  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     288 29.676294408  10.0.0.3        10.255.255.255    ICMP          51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
```

Figure 8: Broadcast example 3: servers reached


*12 interfaces
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

```
eth.src==00:00:00:00:00:03 and not (frame.interface_name=="sw3-eth4" or  frame.interface_name=="sw3-eth5" or frame.interface_name=="sw4-eth4" or frame.interfa
No.      Time          Source          Destination       Protocol  Length Info
```

Figure 9: Broadcast example 3: all other interfaces not reached