Robust Overlay Network with Self-Adaptive Topology: The Reliable File Storage Layer

Loïc Baud

Institut TELECOM, TELECOM ParisTech & LTCI CNRS UMR 5141
46 rue Barrault, 75634 Paris Cedex 13, France
baud@telecom-paristech.fr

***Abstract*—A reliable storage system is a system that ensures the protection of data from loss. Such systems are needed in many critical fields. We cannot imagine that a financial institution looses data about one its clients. Moreover the exponential growing volumes of the data and its availability are crucial issues. A centralized storage system would be overloaded and be a single point of failure. That is why overlay network may provide the solution as they can be used to implement a distributed storage system over an information system. ROSA is an overlay network that was designed to combine a high scalability while maintaining a strong connectivity. The reliability of its topology, designs ROSA as a good platform for deploying such a reliable storage system. Nevertheless ROSA is an unstructured overlay network, this will decrease the efficiency of the storage system. In this article we first propose a structure, called chain of lumps, built upon ROSA. This structure will provide the advantage of a structured network to ROSA. Once ROSA endowed with the structure, we propose a reliable storage system deployed over ROSA. Consequently this system will benefit of the reliability of ROSA.**

I. Introduction

A reliable storage system is a system that ensures the protection of data from loss. Such systems are needed in many critical fields. We cannot imagine that a financial institution looses data about one its clients. Moreover the exponential growing volumes of the data and its availability are crucial issues. A centralized storage system would be overloaded as well as a single point of failure. That is why overlay network may provide the solution as they can be used to implement a distributed storage system over an information system.

The ease of deployment of the overlay networks led them to become, in few years, a solution to many distributed applications. Overlay networks are used for multicast [1], CPU cycle sharing [2] and file sharing (file indexing) [3], security [4][5][6] and QoS management [7][8].

We can distinguish two types of overlay networks, the structured and the unstructured ones. The virtual links of the structured overlay networks are created to form a structure. This structure can be used to build a DHT over the overlay network. A DHT confers to the network some interesting properties. One of these properties is that any nodes are able send a message to any other nodes of the network within a bounded number of hops.

Nevertheless the maintenance of such structures generates overhead. Moreover the constraints on the creation of the virtual links that guaranteed the structure limits the flexibility and the reactivity of the structured overlay networks. Kademlia [9] is an example of structured overlay networks.

The reactivity of unstructured overlay networks is not confined by such constraints. In presence of the failures of nodes or in the case of attack of the network, they reorganize themselves more quickly than the structured ones. Unstructured overlay networks also generate less overhead for their inner operating than the structured ones. Gia [10] is an example of unstructured overlay networks.

It could be interesting for an overlay network to possess the advantages of structured and unstructured ones. In this article we present a way to transform the unstructured overlay network ROSA [11] into a structured one. ROSA is an overlay network that was designed to combine a high scalability while maintaining a strong connectivity. But ROSA is an unstructured overlay network. We will endow ROSA with a structure and therefore the new overlay network obtained will have the benefits of ROSA and those of a structured network.

Many overlay networks are dedicated to storage facility such as Past [12] and Oceanstore [13]. More sophisticated overlay networks implements complete file systems such CFS [14], Ivy [15], Farsite [16]. But none of these overlay networks possess a reliable topology, the file storage system implemented on ROSA endowed by the structure will have it.

II. ROSA

A. The DESEREC project

ROSA is a part of the DESEREC project[1]. A goal of the DESEREC project is to build a secure system that is able to guarantee that it will remain secure over time. An application that detects anomalies, maintains and improves overall security is deployed upon the overlay network ROSA. The application consists in a distributed set of sensors that provide normalized measures used to compute Security Assurance (SA) values [17]. Afterwards, ROSA implements a Security Policy based of these computed SA values. A detailed description of this tool can be found in [18].

B. ROSA

We have built ROSA in such a way that its topology is highly scalable and possess a strong connectivity. Consequently, ROSA has to dynamically reorganize the neighbors set of each node according to:

[1]IST FP6 DESEREC Project (CN: 026600) of the European Union.

- the topology of the underlying network ;
- the maximum number of neighbors of a node ;
- the failures of the elements of the virtual links.

Since there is no structure to maintain, only the preservation of the strong connectivity imposes constraints on the virtual links.

In ROSA, a lump is a set of fully virtually connected nodes. The lumps are the main components of this overlay network. The nodes composing the lumps are called members. ROSA can be seen as an entanglement of lumps. In Figure 1, a small overlay network is schematized on the left and, on the right, the three corresponding lumps are shown.

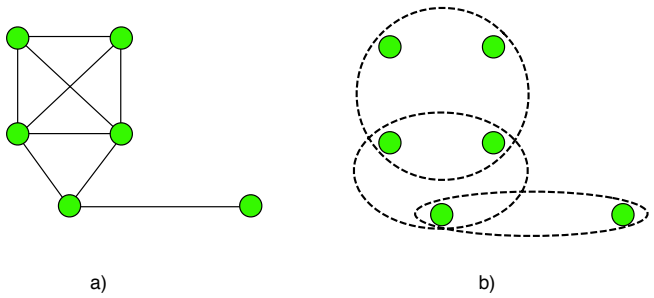

Fig. 1: a) An overlay network b) The 3 *lumps* of this network.

Each node in ROSA has an identifier, a neighbor's set and a lumps set. The identifier is obtained by hashing the IP address of the node. The neighbor's set of a node is the set of the identifiers of its neighbors. The lumps set of a node is the set of the lumps to which the node is a member. At the origin, a lump is represented in the memory of a node by the list of the member's identifiers and the list of the virtual links. Once ROSA endowed with the chain of lumps structure, some additional information (the list of the sub-intervals of I_{init}) will be added to the lump description.

The density of a lump is the minimum number of failures of the virtual link that are necessary to isolate a node from the other nodes. It consists in finding the minimal vertex cut set of the graph of the underlay elements composing the virtual links. In the current implementation, we use the solution proposed in [19] to compute this minimal vertex cut set. The protocol of ROSA ensures that any member of a lump is able to compute its density.

The ROSA principle can be compared to a pancake mixture. A good mixture must have a homogeneous density. The big lumps have to be diluted in order to increase the density of area with less flour. The same idea is applied to ROSA, the nodes leave neighbors, splitting lumps with high density to join and increase the density of lumps with poorer densities. A complete description of the ROSA overlay network protocol can be found in [11].

C. ROSA and lumps modifications

The ROSA protocol consists for nodes in creating and breaking virtual links with other nodes in order to increase the densities of the lumps that have a poor density while homogenizing the densities of the lumps. According to this protocol, lumps may be absorbed by bigger ones or split in two smaller lumps. We explain in this section how and when these modifications are performed.

A lump is absorbed by another lump when all its members are also members of a bigger one. It eventually happens when nodes join a lump or when a lump is split. The Figure 2 and Figure 3 show the two cases of lump absorptions.

In the first figure, the node n joins the lump l to form a new lump called l_1. The set of the members of this new lump include all the members of the lump $l^{'}$. Therefore the lump $l^{'}$ is absorbed by the lump l_1.

In the second figure, the virtual link between the nodes m_1 and m_2 is broken and therefore the lump l is split into two new lumps l_1 and l_2. The set of the members of l_2 is included in the set of the members of $l^{'}$. The lump l_2 is absorbed by the lump $l^{'}$.

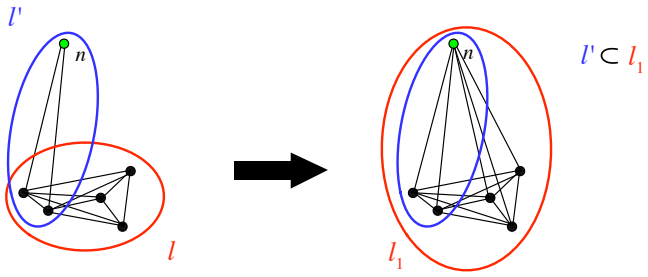

Fig. 2: Absorption after a lump join

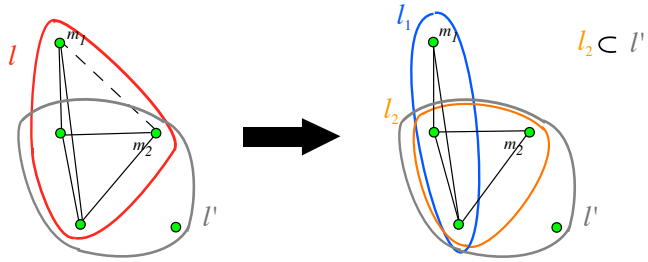

Fig. 3: Absorption after a lump split

III. The chain of lumps structure

A. Description

The structure is composed of all the lumps. The lumps are organized as a chain. It implies that all the lumps are at least a link of this chain. To build such a structure lumps, we define first the initial interval I_{init} as: $I_{init} = [0, 2^{64} - 1]$. Then we assign to each lumps at least one sub-intervals of I_{init} in such a way that:

- all the sub-intervals are disjoints;
- the union of all the sub-intervals is equal to I_{init};
- two lumps that possess contiguous sub-intervals of I_{init} have at least one member in common.

Let consider a lump l that has only one sub-interval I of I_{init}, let the lump possessing the sub-interval just after I be

the successor of l and the lump possessing the sub-interval just before I be the predecessor. When a lump possesses many sub-intervals of I_{init} it has as many successors and predecessors than the number of sub-intervals that it owns.

The projection of the whole network onto an integer interval combined with the facts that any partition of this interval is allocated to a lump and the fact that any lump has a member in common with its predecessors and its successors ensures that the algorithm proposed further in this article will terminate. The projection onto a greater dimensional space would allow for better routing quality, but the construction and maintenance of the resulting structure would have been much more complex. Consequently the gain made at the cost of routing auait been lost by the increased cost of maintenance.

A node may be member of many lumps and a lump may possess more than one sub-interval of I_{init}. Consequently, there are short cuts in the chain of lumps. The chain of lumps structure can be schematized as in the Figure 4. In the this

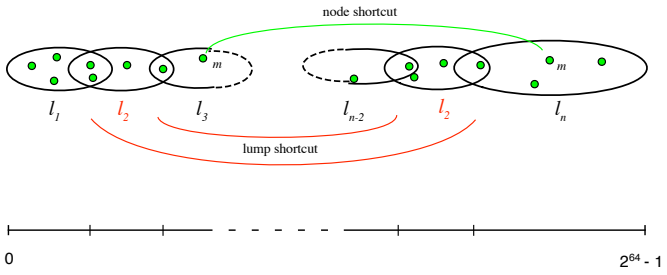

Fig. 4: The chain of lumps structure

figure the chain of lumps is represented. One can see that the node m appears in two different lumps, creating a node short cut. The lump l_2, has two sub-intervals of I_{init} , therefore it appears twice in the chain of lumps, creating a lump short cut.

B. Construction and maintenance

When the overlay network is created, there is a single lump and, this one except, lumps are created only during the split of lumps. To initiate the chain of lumps structure the initial interval I_{init} is given to the first lump. We will see in the following sections how the structure evolves according to the splits and absorptions of lumps. These short cuts will reduce the number of hops needed to route messages from nodes to lumps.

1) Splits: Let l be a lump that owns only one sub-interval $I = [a, b]$ of I_{init}. As schematized in Figure 5, let l_{left} and l_{right} be the predecessor and the successor of l respectively. We will see in Section III-C that deals with the operating proof of the structure that if l splits into two new lumps, l_1 and l_2, and if $l_{left} \cap l \neq l \cap l_{right}$ or $\#(l_{left} \cap l) > 1$ or $\#(l \cap l_{right}) > 1$ then:

- either 1) $l_1 \cap l_{left} \neq \emptyset$ and $l_2 \cap l_{right} \neq \emptyset$;
- and/or 2) $l_2 \cap l_{left} \neq \emptyset$ and $l_1 \cap l_{right} \neq \emptyset$.

If we consider the case of the Figure 5, $I_1 = [a, \lfloor (a+b)/2 \rfloor]$ and $I_2 = [\lfloor (a+b)/2 \rfloor, b]$. I_1 will

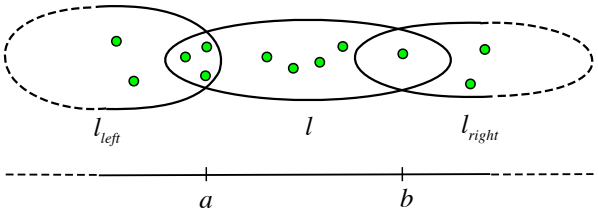

Fig. 5: Example of split process (1).

be given to l_1 and I_2 to l_2 as schematized in Figure 6. If we consider the other case I_1 will be given to l_2 and I_2 to l_1. In the case where 1) and 2) are true, I_1 will be given

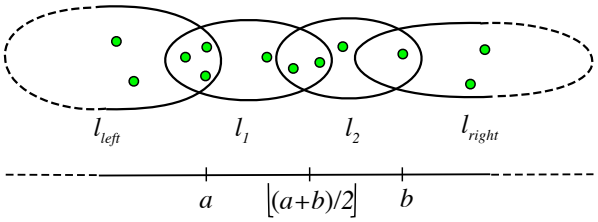

Fig. 6: Example of split process (2).

to l_1 and I_2 to l_2 if $min(\#(l_{left} \cap l_1), \#(l_2 \cap l_{right})) > min(\#(l_{left} \cap l_2), \#(l_1 \cap l_{right}))$ else I_1 will be confided to l_2 and I_2 to l_1. This ensures that the size of the intersections between the lumps of the chain is maximized.

If a lump that possesses more than one sub-interval of I_{init} the same process is repeated for each sub-interval.

2) Absorptions: When a lump absorbs another lump, it becomes the owner of the sub-intervals of I_{init} of the absorbed lump. If a lump absorbs one of its predecessor or successor, then it possesses contiguous sub-intervals. Therefore, these contiguous sub-intervals are merged. An absorption process is described in Figure 7. In this figure, the lump l_2 appears

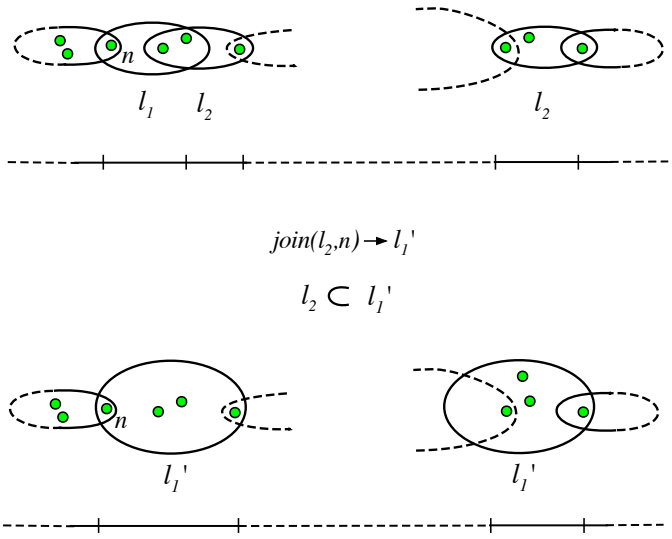

Fig. 7: Example of absorption process.

twice in the chain of lumps structure. The node n joins l_2 to

form the lump $l_1^{'}$. The lump l_2 is absorbed by the lump $l_1^{'}$.

C. Proof of concept

We will show, in this section, that despite of the modifications of the lumps the structure is maintained. It is obvious that the absorption of lump is not a problem. Nevertheless, with regard to the splits of the lumps we have to deal with several cases.

Unless there is too many underlay elements failures at the same time, ROSA ensures that no lump with a density equals to 2 will be split. Therefore we will only consider lumps with at least three members. To facilitate the understanding of the proof, we also only consider the lumps that have a single sub-interval of I_{init}. The proof principle stills the same if we consider lumps with more than one sub-interval.

Let us consider the split of a lump l, with l_{left} as its predecessor and l_{right} as its successor. This split generates two lumps l_1, l_2. The structure will be maintained if one the following conditions is satisfied:

- either $l_{left} \cap l_1 \neq \emptyset$ and $l_2 \cap l_{right} \neq \emptyset$;
- or $l_{left} \cap l_2 \neq \emptyset$ and $l_1 \cap l_{right} \neq \emptyset$.

In some minor cases it is not always possible to satisfy this condition and we will see in the following how these "wrong case" are handled.

1) The good cases: If $l_{left} \cap l \neq l \cap l_{right}$ or $\#(l_{left} \cap l) > 1$ or $\#(l \cap l_{right}) > 1$ the condition is always satisfied.

Proof: Let us suppose that $l_{left} \cap l \neq l \cap l_{right}$ or $\#(l_{left} \cap l) > 1$ or $\#(l \cap l_{right}) > 1$. It means that $\#((l_{left} \cap l) \cup (l \cap l_{right})) \geq 2$. In other words: $\exists\ m_{left} \in (l_{left} \cap l)$ and $m_{right} \in (l \cap l_{right})$ such $m_{left} \neq m_{right}$.

According to the ROSA protocol, the split of l generate two lumps l_1, l_2 such that: $l_1 = l \setminus \{m_2\}$ and $l_2 = l \setminus \{m_1\}$. Then, if m_{left} and m_{right} are both different of m_1 and m_2 it is obvious that the condition is satisfied.

Moreover, if $m_{left} = m_1$ ($m_{left} = m_2$) then $m_{left} \in l_1$ ($m_{left} \in l_2$) and according to the definition $m_{right} \in l_2$ ($m_{right} \in l_1$) and the condition is satisfied too. The same argument can be applied to m_{right}. ∎

2) The wrong case: $l_{left} = l_{right}$ and $\#l_{left} = 1$: In the case of $l_{left} = l_{right}$ and $\#l_{left} = 1$ it may happen that the condition is not satisfied. That means that one of the two lumps resulting from the split has no member in common with l_{left} and l_{right} while the other one has a member in common with both (see Figure 8).

In this case, the sub-interval of I_{init} is given to the lump that has a member in common with l_{left} and l_{right}. The chain of lumps structure can be finally schematized as in Figure 9.

We have to add a new rule to the ROSA protocol in order to avoid the propagation of lumps with no sub-interval. This rule is the following: *Lumps with no sub-interval cannot be joined by any node except if this join leads to the absorption of this lump.*

D. Routing from node to lump with the structure

The chain of lumps can be used to route messages from nodes to lumps. Let $K \in I_{init}$ be a key, let I_{target} be

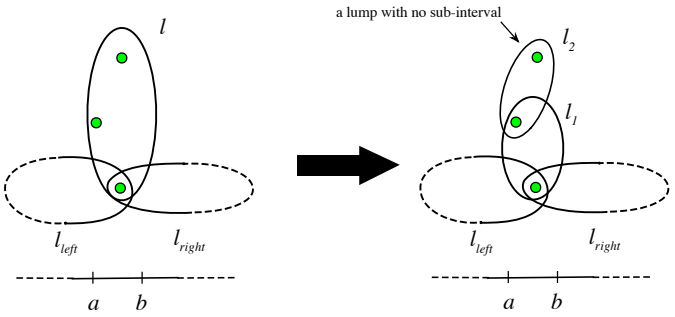

Fig. 8: Example of a wrong case

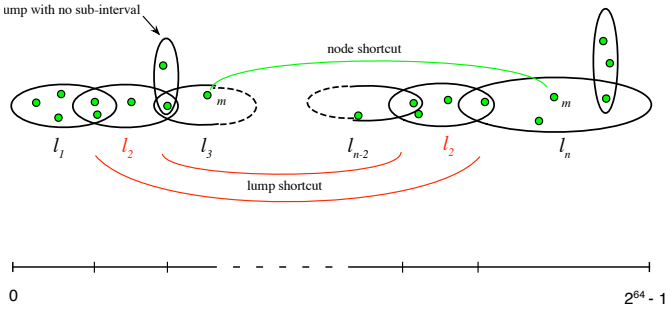

Fig. 9: The chain of lumps structure

the sub-interval containing K and let L_{target} be the lump possessing I_{target}. A node that wishes to send a message M to L_{target} sends messages getPart(K) to its neighbors. When a node receives such a message, it replies with a message closestPart(R). R is the closest bound to K among the bounds of the sub-intervals owned by the lumps to which this node belongs. The node that wants to send the message M receives many messages closePart(R). Eventually this node sends a message propagateMessage(M,K) to the neighbor that replied with the closest R to K.

This process is repeated until a node belonging to L_{target} was reached. One step of this process is schematized in Figure 10. In this Figure, we deliberately ignore lumps short cuts, nodes short cuts and the lumps with no sub-interval.

In the case where this is a node that only belongs to lumps with no sub-intervals of I_{init}, the message M is broadcasted from neighbors to neighbors until it reaches a node belonging to a lump in the structure. Once the structure reached, the process described above is used.

To validate this routing algorithm, we have performed few simulations. In the first one we have generated a chain of lumps structure composed of lumps with two members only. Each lump possessed only one sub-interval of I_{init}. This is the worst case for our routing algorithm. On the contrary, since each node had only two neighbors (excepted two of them that have only one neighbor), it is very favorable case for a flooding algorithm. Then, we measured the number of messages generated to route a message from a node to a lump. The Figure 11 shows the number of messages sent

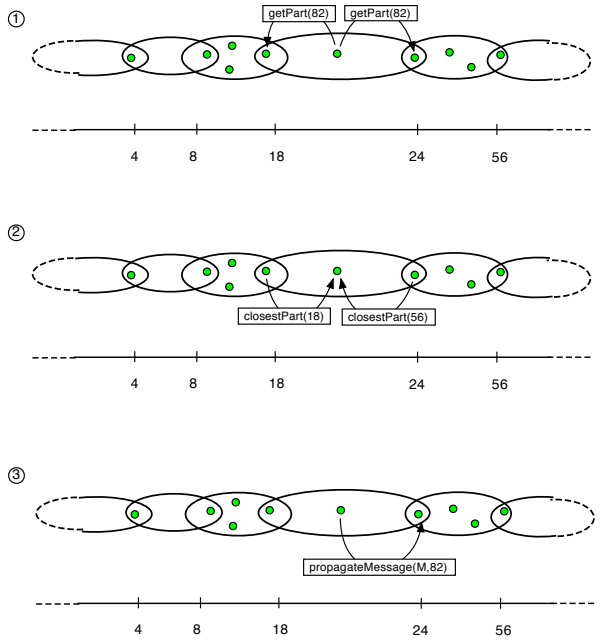

Fig. 10: A node wants to send a message M to the lump that owns the part containing 82

by a flooding algorithm and by our algorithm depending of the number of lumps of the structure. In this experiment, we voluntarily ignore the getPart and closestPart messages since the size of these ones is only few bytes and therefore negligible. The simulation shows that our algorithm is better than a flooding one, even in the worst conditions.

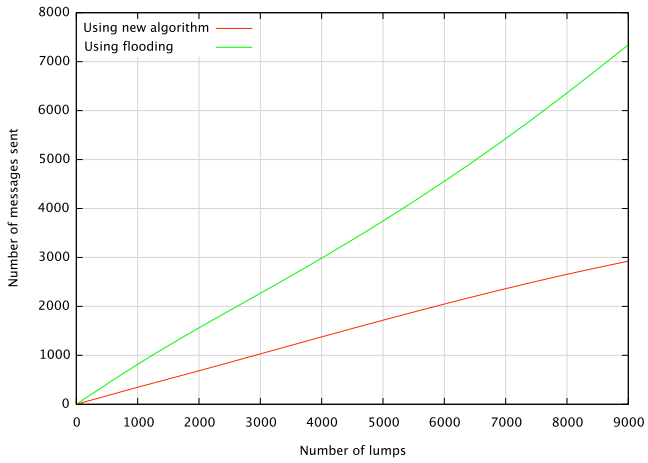

Fig. 11: Comparison between a flooding algorithm and the proposed algorithm.

The next experiments show the impacts on the efficiency of the lumps short cuts (see Figure 12) and nodes short cuts (see Figure 13). The number of messages sent decreases as the number of short cuts increases. The chain of lumps structure used for these two simulations was composed of 100 lumps of 20 nodes each. These two experiments confirm that the conditions of the first one were the worst since there was no short cut. Then, we can affirm that the proposed algorithm is a lot better than the previous flooding one.

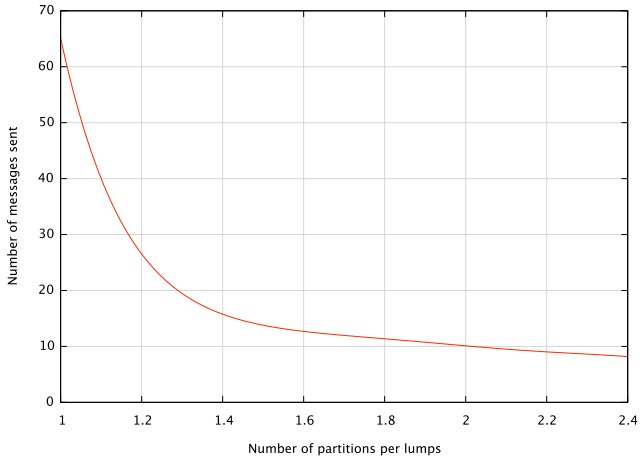

Fig. 12: Impact of the lumps short cuts on the number of message sent.

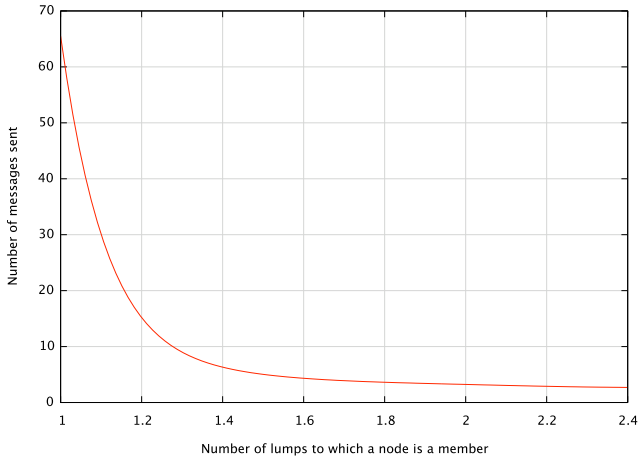

Fig. 13: Impact of the nodes short cuts on the number of message sent.

IV. The File Storage Layer

A. Files indexes description

Each file stored on ROSA is stored in many replicas. Each replica is stored by a node. The number of replicas is a parameter that has to be decided when the file is stored. To help finding these replicas, an index is also stored on the network. This index contains, the file identifier, some data about the file, a flag, the lists of the locations of the replicas and the list of the identifiers of the nodes that owns this file. A file index is described in Figure 14. The file identifier is used to locate the file over the structure. The data is simply the file name and a short description. The flag is used to specify that the file is being updated or deleted. The version of the file

File ID	Version	Flag
ID_{file}	Version nb	flag
File data		
File name, description, etc.		
Replicas		
$ID_{node1} : [a_1, b_1]; [a_2, b_2]; ...$		
$ID_{node2} : [a_3, b_3]; ...$		
...		
Owners ID		
ID_{owner1}		
ID_{owner2}		
...		

Fig. 14: A file index

is an integer. A location of a replica is a node identifier and the sub-intervals of the lumps to which the node is a member. The list of the owner identifiers is the list of the identifiers of the nodes that are allowed to access, update and delete the file. The index of each file is stored on all the members of the lump that possesses the sub-interval owning the file identifier. This lump will be called the lump index of this file.

B. Storing a file

A node that wants to store a file upon ROSA, computes the hash of the name and the size of the file to obtain the file identifier, then the node determines the number of replicas needed and which nodes are allowed to access to the file. Afterwards, the node builds a StrFile message. This message, as described in Figure 15, contains the sending node identifier and location, the file identifier, an optional description of the file, the number of replicas wanted, the list of the owners and the file itself. We call node location the list of the sub-intervals of the lumps to which the node is a member. This node location will be useful if a node needs to reply to this message. Eventually this message is sent to the lump with the sub-interval owning the file identifier using the process described in Section III-D. This lump will be the lump index of this file. When a node of the lump index receives a message

StrFile	
Node ID	Node location
File ID	File description
Rep. nb	Owners
File	

Fig. 15: A StrFile message

StrFile, if a file with the same identifier is already stored, the node replies with a message BadID. Else, it builds and sends a message StrIdx to the members of the lump index. These members will store the index of this file. The message StrIdx contains the index of this file where the Flag is set to 'free' and the Version to 1.

The node that receives the message StrFile also randomly chooses N elements of I_{init}, where N is the number of replicas wanted. Then, it builds a message ScttrRep and sends it to the lumps possessing the sub-intervals that own the chosen elements, using the process described in Section III-D.

A node of the targeted lump receiving such a message sends a message StrRep to a randomly selected member of this lump. This message informs the receiving node that it has to store a replica of the file. ScttrRep and StrRep messages contain both the file identifier and a copy of the file.

C. Retrieving, updating and deleting a file

Only the owners of a file are allowed to retrieve, update and delete it and the necessary condition in order to do it is to be aware of the file identifier.

1) Retrieving: When a node wants to retrieve a file, it sends a message RtrvFile to the lump index of the file. A message RtrvFile is described in Figure 16. When a node of this lump

RtrvFile	
Node ID	Node location
File ID	

Fig. 16: A RtrvFile message

receives such a message, it replies by a message NotAllwd if the sender is not an owner in the file and by a message Wait if the flag in the index is not set to 'free'. In the other cases, it replies with a message RepLoc containing the replica locations and the current file version. The node willing to retrieve the file has to send a message RtrvRep to any node storing a replica of the file. This one will reply by a message HereItIs containing the file and the version number.

The function of the messages Wait is to inform the node that wants to retrieve the file that it has to wait an indefinite amount of time before receiving the message RepLoc. We will see in the next sections in which cases it happens.

2) Updating: To update a file, a node owner has first to retrieve the file. Then it has to send a message UpdtFile to the lump index of the file. This message is described in Figure 17. The file version must be the number obtained when retrieving

UpdtFile	
Node ID	Node location
File ID	File version
Changes	

Fig. 17: A UpdtFile message

the file. The changes are the differences between the retrieved file and the new one. These differences are obtained using the differential file comparison utility diff [20].

When a node of the lump index receives such a message it checks many things:

- that the file exists ;

- that the sender is an owner of the file ;
- that the flag in the index is set to 'free' ;
- that the version number is the same that the one if the index.

If one of these conditions is not respected, it replies by a message FileNotFnd in the first case, NotAllwd in the second case and FileMdfd in the other cases. If all the conditions are respected, the node has to send messages FileBusy to the members of the lump index of the file. This message contains the file identifier. Each node receiving this message set the flag in the corresponding index to 'busy'.

Afterwards the node sends a message UpdtRep to the node storing the replicas. This message contains the file identifier and the changes to apply to the file. At the receipt of this message, the nodes storing the replicas sends an ACKUpdt to lump index. When all the ACKUpdt are received the members of the lump index set the flag of the corresponding index to 'free'. If after a determined amount of time an ACKUpdt still missing, the replica is declared lost and a node replica substitution procedure is engaged. This procedure will be described in Section IV-D3.

3) Deleting: To delete a file, a node owner has to send a message DltFile to the lump index of the file. The member of the lump index that receives this message first checks if the sender is indeed an owner. If not, it replies by a message NotAllwd. If the sender is an owner, the member sends a message DltIdx to the other members of this lump. At the receipt of this message the members deletes the corresponding index.

The member that receives the message DltFile also has to notice the nodes that store the replicas about the deletion of the file. Sending a message DltRep to them does it. The node receiving this message deletes their replicas of the file. Messages DltIdx and DltRep only contain the node identifier. To finish the member that has received the message DltFile deletes his own copy of the index.

D. Preservation of indexes and stored files

1) Preservation of indexes: As shown in the previous sections, lumps can be split or absorbed by other lumps. Since the index of a file is stored by all the members of a lump, the modifications of the lumps affect the indexes.

When a lump is split, and therefore some sub-intervals are split too, the indexes have to be redistributed. Since after a split the lumps set of the nodes are modified, it may happen that a node store the index of a file when it is not a member of the lump index of this file anymore. In this case this node has to discard this index.

When a lump is absorbed, indexes have to be redistributed too. When a lump is absorbed, some nodes become members of lumps that possess sub-intervals that were not possessed by the lumps of their old lumps set. The corresponding indexes have to be forwarded to these members by the other members.

2) Preservation of stored files: The nodes storing the replicas of the files may have failure or leave the network, therefore the number of replicas requested for a file may not be maintain. To compensate for these losses, axs mechanism has to be implemented. It consists for the nodes storing the replicas in periodically sending message RepAlive to the lumps index of the corresponding files. This message contains the sender identifier, the sender location and the file identifier.

Once a member of the lump index receives this message, it checks in the index if the node sender is one of the nodes storing the replica of the file. If this condition is respected, it replies by a message ACKRepAlive. Afterwards, it forwards the message RepAlive to the other members of the lump. These members do not have to send ACKRepAlive.

If, after a determined amount of time, a node storing a replica did not send message RepAlive, a node replica substitution procedure is engaged. It may also happen that a node storing a replica did not receive the message DltRep. In this case, it may happen that a node stores a file whereas it does not have to do anymore. To prevent that waste files encumbers the network, the nodes storing the replicas, which have not received a message ACKRepAlive since a determined amount of time have to delete their replicas.

3) The node replica substitution procedure : If a node storing a replica does not periodically send messages RepAlive or does not reply to a message ACKUpdt, it has to be substituted. Only a member of the lump index of a given file can start a node replica substitution procedure.

To start such a procedure, a member first checks in the index of the corresponding file that the flag is set to 'free'. If the flag is set to 'busy' the procedure is cancelled. Else the member sends a message FileBusy to the members of the lump index. Afterward, it randomly chooses an element of I_{init}, retrieve a copy of the file using the process described in Section IV-C1. Then, it sends a message ScttrRep, containing the file identifier and a copy of the file, to the lump possessing the sub-intervals that owns the chosen element. Each node receiving this message set the flag in the index to 'busy'. The messages ScttrRep is handled as described at the end of the Section IV-B.

V. Conclusion

This article proposes an enhancement of the overlay network presented in [11]. And without modifying the initial protocol, endows ROSA with a structure. The result is a structured overlay network with the same interesting properties, strong connectivity and high scalability, as the old ROSA.

We also proposed an algorithm that ensures that every node of the overlay network can send a message to any lump within a small amount of hops. We have performed some simulations to demonstrate the efficiency of this algorithm. These experiments shown that our algorithm is a lot better that the previous flooding one, even in the worst condition. However, mathematical validation and studies remain to do. Using this new algorithm and the chain of lumps structure we have also developed a file storage system. This system can be considered reliable since it was deployed over a reliable overlay network. This system is also said to be reliable since each file is stored in many replicas in different location of the

overlay network and since a duplication mechanism assures that if a replica is lost another one will be created and stored. This way, if failures occur, the files stored over the network are preserved. A problem subsists with this system, a malicious node may perform some denial of service attack. Without authentication, we have to assume that all the nodes are have a correct behavior.

Concerning ROSA, our future work will consist in building a CPU sharing cycle systems using the structure presented in this article. Another axis of research, concerning the structure, consists in finding a way to suppress the lumps that do not own sub-interval of I_{init}.

REFERENCES

[1] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. Almi: An application level multicast infrastructure. pages 49–60, 2001.

[2] D. P. Anderson. Boinc: a system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing.*, pages 4–10, 2004.

[3] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 53–65, 2002.

[4] Guofei Gu, Prahlad Fogla, Wenke Lee, and Douglas Blough. Dso: Dependable signing overlay. In *Proceedings of The 4th International Conference on Applied Cryptography and Network Security (ACNS '06)*, 2006.

[5] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global intrusion detection in the domino overlay system. In *In Proceedings of Network and Distributed System Security Symposium (NDSS*, 2004.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *In Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[7] David G. Andersen, Hari Balakrishnan, and G. Andersen. Resilient overlay networks. In *Symposium on Operating Systems Principles*, pages 131–145, 2001.

[8] Sameer Qazi and Tim Moors. Scalable resilient overlay networks using destination-guided detouring. In *Proceedings of the IEEE International Conference on Communications (ICC)"*, 2007.

[9] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.

[10] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, 2003.

[11] Loic Baud, Nguyen Pham, and Patrick Bellot. Robust overlay network with self-adaptive topology: Protocol description. In *Proceedings of 7th IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*, pages 154–160, 2008.

[12] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.

[13] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geeis, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, pages 190–201, 2000.

[14] Frank Dabek, Frans M. Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, 2001.

[15] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. *SIGOPS Oper. Syst. Rev.*, 36:31–44, 2002.

[16] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI*, pages 1–14, 2002.

[17] Nguyen Pham and Michel Riguidel. Security assurance aggregation for it infrastructures. In *ICSNC '07: Proceedings of the Second International Conference on Systems and Networks Communications*, page 72. IEEE Computer Society, 2007.

[18] Nguyen Pham, Loic Baud, Patrick Bellot, and Michel Riguidel. Towards a security cockpit. *isa*, 0:374–379, 2008.

[19] C. Patvardhan, V.C. Prasad, and V.P. Pyara. Vertex cutsets of undirected graphs. *IEEE Transactions on Reliability*, 44:347–353, 1995.

[20] M.D. Mcllroy J.W. Hunt. An algorithm for differential file comparison. Technical Report Computing Science Technical Report No.41, Bell Telephone Laboratories, 1976.