Fast reconfiguration of dynamic networks

Bernard Cousin, and Miklos Molnar

***Abstract*— To prevent from topology changes, failures, or overloads, network management requires frequently computation and reconfiguration of established connections. When the number of established connections in the network is very large, the optimization of the reconfiguration task is essential to have short latency. When the communications using these connections are real-time, traffic interruption is not acceptable, and thus the scheduling of the reconfiguration tasks can be difficult. In this paper, we propose a method which reconfigures unicast connections efficiently and without connection break. Our simulations show that the reconfiguration time requires by our algorithm is lower than usual reconfiguration methods, and scales well with the number of nodes in the network.**

***Index Terms*— Network, connection management, path establishment, router configuration, scheduling algorithm.**

I. Introduction

Today's networks have to deal with Qos, load balancing, survivability. All these enhanced services require close management of the network and thus frequent reconfigurations of the network (cf. [1], [11]).

In connection-oriented networks, when an establishment of a connection is requested, based on the current network topology, the current load of the network and the domain policies (i.e. load-balancing policy, path protection policy, etc.), paths which fit the QoS parameters of the request are computed (cf. [8]). Once the path computed, the nodes on the path have to be configured, for instance to update the forwarding table, to modify the parameters used to control the queuing discipline or to shape the traffic, etc. Later, because of changes in the topology, in the network load or even in the network policies, the paths used by the connections have to be recomputed and nodes on the new and old paths reconfigured.

The connection management process of a network could be seen as a control loop between network control entities and forwarding entities (cf. Fig. 1). The control loop as three phases: data collection, path computation and router configuration. Data collection (for instance, topology change notifications, load measurements, connection requests, etc.) flows from the network (network nodes or application hosts) to the control entities. The control entities will react to these data in computing new paths for some connections. Then the forwarding entities (the routers) on the new and old paths have to be reconfigured.

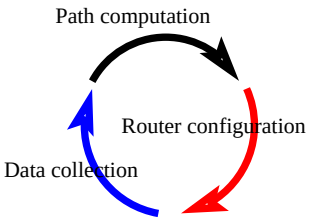

Fig. 1. Management process of connections

Due to the distributed nature inherent to a network and to preserve the connection from the source to the destination, path configuration should be done in a coordinated way. For instance, appropriated configuration messages have to be sent from the control entities which have computed the new path toward all the forwarding entities on the path. In the following chapter we will introduce in more details the objectives and the required coordination. But let's give one example of coordination requirements. For instance as we do not want the connections to be interrupted, the new path should be setup before the old path is removed (usually this method is called *make before break*). That will lead to an order in the sending and processing of the configuration messages. Our paper will address this point, trying to minimize the delay of the reconfiguration process.

Many protocols have been proposed to establish paths. For instance for MPLS network, LDP (Label Distributed Protocol) [2] or RSVP are used to set up LSRs. GSMP (General Switch Management Protocol) provides switch configuration control and reporting for ATM, Ethernet, MPLS, TDM or optical switches [3]. Most of these protocols dictate one special order in the node configuration, generally the order of the node on the path (or its reverse order). We will show that the total order which is induced by these protocols is not optimal.

Anyhow all these protocols do not deal with reconfiguration (updating of a path) and do not take advantage of the induced optimization. The usual way they propose to manage reconfiguration is to tear down the old connection and to set up a new one. First, due to the restrictive properties of some network and given in the next paragraph, it's not always possible to setup the new connection before the old one. So the traffic is interrupted until the new connection is established. Second, even if it's possible, this usual way do not

B. Cousin is with the Department of Computer Science, University of Rennes 1, France (phone: +33 299 84 73 33; fax: +33 299 84 71 71; e-mail: Bernard.Cousin@ irisa.fr).

M. Molnar is with INSA Rennes (e-mail: Miklos.Molnar@irisa.fr).

benefit from the fact that some segments of the paths of the old and new connection could be shared, and thus should not need to be torn down and then reestablished.

In some kind of networks the connection path could have some restrictive properties which require close coordination. For instance, close coordination is required when the packet forwarding is determined by an ID global over the whole network (e.g. a destination address). Then, packets on the new path could not be distinguished from packets on the old path (they have the same ID). Thus the configuration of a node for the new path has an impact on the old path. This case happens in networks, like Internet, where there exists at most one entry (i.e. one next hop) in the forwarding table for each unicast destination address.

The burden and the complexity of the configuration must not be neglected. Configuration could be done on a very frequent basis. The number of messages sent between control and forwarding entities could be large and become larger as the network size and number of connections increases (cf. [5]). Furthermore, one well known problem in network management (like in any close loop control system) may arise. This problem happens because, when new paths are computed and reconfigured, the computation is based on data gathered from past network status but the path will be set up after a certain inevitable delay (data gathering, path computation, configuration message transmission, etc). If the behavior of the network changes faster than the reconfiguration delay the setting could be dramatically out of phase. One way to solve the swinging is to reduce the latency between the events which have launched the computation of a new path and the effective configuration of the nodes on the new computed path. To have an efficient reconfiguration process will enable low latency and better network resource utilization.

Not every network is connection oriented, nevertheless there is a tendency to introduce in many networks the notion of (pseudo-)connection. This connection notion is required to add new services like VPN, QoS, route protection and explicit routing. For instance, when you want to deal with QoS, or to give some guarantee about the data transmission service or to make some efficient traffic engineering you have to manage at least soft-states into the network nodes. One actual way to offer QoS management (traffic separation) at IP level is to use an under-layering network like MPLS [4]. MPLS uses the concept of connection; they are called LSPs (label switch paths). For another instance, IP through RSVP uses soft-states. And packet scheduling in RSVP routers should be configured in accordance with the QoS policies and the path selected at the edge route of the RSVP domain.

Our work could be applied to all kinds of network: WDM or optic networks use light paths; IPV6 with its labels introduces soft-connection into Internet routing level; MPLS (and its generalization GMPLS) uses LPSs.

However if we make a closer look we could see that the same node could belong to the old and the new paths but having different next hops (downstream nodes) or having different upstream nodes. This node has to be configured before some of the nodes and after some others, thus a strict scheduling for node reconfiguration is needed.

II. RECONFIGURATION METHOD

A. Objectives

We assume that in the network there is a connection from the source *S* to the destination *D*. This connection uses a path *Old*. The configuration process swaps the path *Old* for path *New*. The objectives of the reconfiguration process are two folds:

- The reconfiguration process should not interrupt the connection,
- The reconfiguration process should be as fast as possible.

To reduce the duration of the reconfiguration process we can try to reduce either the number of nodes to be reconfigured, or the delay for each node configuration, or the number of steps in the reconfiguration process. The minimal list of nodes to be reconfigured is easy to determine. It is the union of the nodes in *Old* and *New* paths minus the nodes which have the same next hop on the two paths. Thus we cannot reduce this more. The delay for each node configuration includes the transmission delay of the configuration message between the control entity and the node to be configured, the duration of the reconfiguration itself at the node, and the transmission of the acknowledgment message. We suppose that none of these delays can be reduced, because they depend on the characteristics of the network and of the processor of the node. In consequence the only way to reduce the duration of the reconfiguration process is to regroup several node configurations into one reconfiguration step. All the node configurations belonging to the same step are executed in parallel. And the steps are scheduled in a sequential manner, one after the other.

In consequence, an optimal reconfiguration method will produce a minimal number of reconfiguration steps where each step contains a set of nodes, and if each node of the minimal list of nodes belongs to only one reconfiguration step. The list of steps determined the order when the configuration of the nodes has to happen. This order assures that no interruption of the connection happens during the overall reconfiguration process.

B. Upstream configuration

One obvious reconfiguration method is to start the reconfiguration from the destination D, following the *New* path upstream, configuring all the nodes on the *New* path one reconfiguration step after the other, each step containing exactly one node. A last step is added to configure all the nodes of the *Old* path which do not belong to the *New* path. It

is easy to prove this upstream method does not interrupt the connection: at each step, there always exists a path between the source and the destination. The number of steps of this upstream method is equal to the number of nodes into the *New* path, minus one because it is never required to configure the destination node, plus one for the last step (which will configure the remaining nodes).

Let assume the network describes in Fig. 2. Suppose that the *Old* path is <S, 2, 3, 4, 5, 6, 7, D>, and the *New* path is <S, 9, 10, 4, 3, 2, 11, 12, 7, D>. The list of the reconfiguration steps produced by the upstream reconfiguration method is: <{7], {12}, {11}, {2}, {3}, {4}, {10}, {9}, {S}, {5, 6}>.

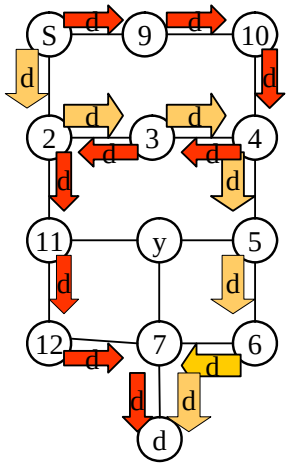

Fig. 2. A network with an old and a new path

C. *Optimized upstream configuration*

The previous method can be optimized, if we introduce the notion of latch. A latch is a node belonging to the two paths but which next hops are different on the two paths. For instance S, 2, 3, and 4 are all the latches of the previous example.

This optimized method proposes to reconfigure in a first step all the nodes which are only on the *New* path. Then the latches on the *New* path are reconfigured moving upstream from the destination, one latch after the other. Finally all the remaining nodes are reconfigured in a last step.

If we assume the previous example, the reconfigurations steps are: <{9, 10, 11, 12}, {2}, {3}, {4}, {S}, {5, 6}>. We should notice that some nodes do not need to be reconfigured because they belong to the two paths and have the same next hop (e.g. node 7) or have no next hop (e.g. D). We call *Old* (resp. *New*) the *Old* path (resp. *New* path) restricted to the nodes which need to be configured.

D. *Our reconfiguration method*

We introduce another concept: isolated nodes. A node is isolated during a given step of the reconfiguration process of a connection iff the node does not belong to the path used during this step (a complete path from the source to the destination should exist at each step because it should not have connection interruption). Since isolated nodes are not used for the transmission of data, all isolated nodes during a step can be reconfigured during that step. Let us notice that, first a node can be isolated for some steps of the reconfiguration process and not isolated during some others, second an isolated node can be a latch.

For instance using the previous network example, before the first reconfiguration step the nodes 9, 10, 11, 12 are isolated; after the last step they are not isolated.

We can describe the problem to solve with a graph as in Fig. 3. It is an example of synthetic graph where only latches are listed. It illustrates the overlapping of the latches which makes difficult first the scheduling of the configuration steps and, second the computation of the latches which could be set in parallel for each step.

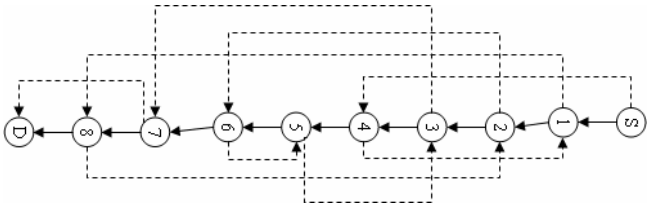

Fig. 3. Dependence graph with overlapping latches

Our reconfiguration method is turn based, and runs until all the nodes are configured. At each turn at least one node is configured, thus the algorithm stops. Each turn has two steps: the first step search for isolated nodes, the second step looks for one appropriate latch. It can be described as:

Until there exists some nodes to be configured
<First step> Configure simultaneously all the isolated nodes with one parallel step.
<Second step> Configure any appropriate latch in one another step.
End

A latch j is appropriate when

$$old(j) < old(suc(j)). \quad (1)$$

After the second step, any node on *Old* between exclusively *j* and *suc(j)* becomes isolated. *old(j)* is the position of the node *j* in the list of the not yet configured nodes in the *Old* path, starting from the source. By definition, *old(S) = 0*. *suc(j)* is the first non configured latch which is the successor of j on *New*. By assumption *suc(j) = D*, if j has no non configured latch successor on *New*.

To accelerate the computation, in our implementation we select the first non configured latch moving upstream on *New* from the destination. By construction this latch is appropriate.

Our algorithm utilizes the following sets and variable:

- *A* is the list of all the connected nodes (I.e. non isolated nodes) to be configured. Initially *A=Old*.
- *B* is the list of all isolated nodes to be configured. Initially *B=New-Old*.
- *C* is the configuration list. Initially, *C={}*.
- *n* is the latch which has been most recently configured. Initially *n =D*.

Until *A* is not empty, we progress upstream on the *New* path using *n* as path location indicator.

If we assume the previous example, our algorithm produces

the following states at the end of each step:

0- A=<S, 2, 3, 4, 5, 6>, B=<9, 10, 11, 12>, C=<>, n=D
1- A=<S, 2, 3, 4, 5, 6>, B=<>, C=<{9, 10, 11, 12}>
2- A=<S>, B=< 3, 4, 5, 6>, C=<{9, 10, 11, 12},{2}>, n=2
3- A=<S>, B=<>, C=<{9, 10, 11, 12},{2},{3, 4, 5, 6}>
4- A=<>, B=<>, C=<{9, 10, 11, 12},{2},{3, 4, 5, 6}, {S}>, n=S

III. EVALUATION

We compare our reconfiguration method versus the upstream reconfiguration method and the optimized reconfiguration method. We used a same set of paths and parameters to compare the three methods. In our simulator, a path generator produces two paths: an old path and a new path. The length of the paths, the rate of node reutilization and, the number of latches shared by the paths can be controlled. By default the rate of node reutilization between the old path and the new one is equal to 0.5.

We have evaluated the number of runs needs to achieve a certain level of confidence. We have computed the mean number of steps over 10, 100, 1000, and 5000 runs on 32-nodes paths. Mean values computed over 100 runs produce a computed error percentage of 1.02 %, whereas 10 runs produce an error of 8.01 %. All our following results will be based on means computed over 100 runs.

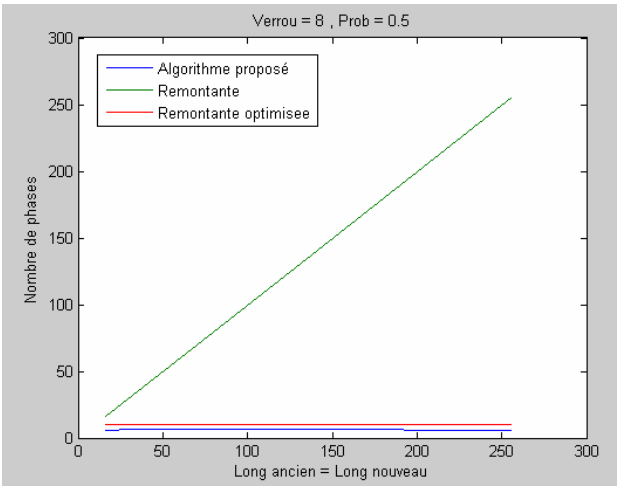

Fig.3. Average number of reconfiguration steps versus path length, the number of latches is 8.

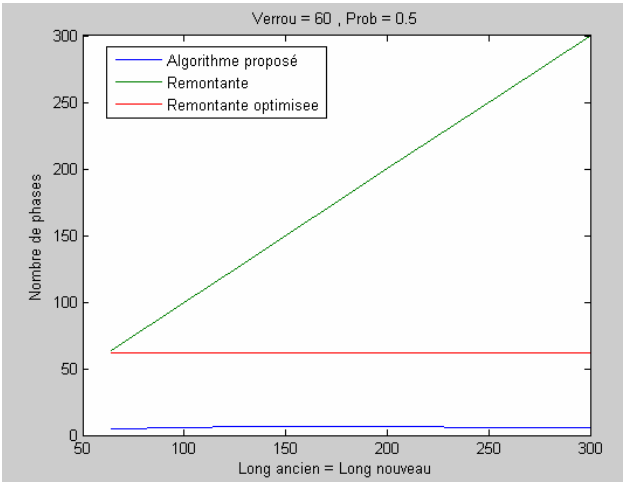

Fig.4. Average number of reconfiguration versus path length, the number of latches is 60.

Figures 3, 4 give strong indications that our reconfiguration method is better than the upstream methods for any path length and any number of latches. The two paths have the same length which varies on the horizontal axis. The vertical axis is the average number of reconfiguration steps. The improvement can be very important. For instance for 32-node paths having 8 latches, our method requires in average 6.48 steps which is an improvement of 45 % versus the optimized upstream method and 80 % versus the upstream method.

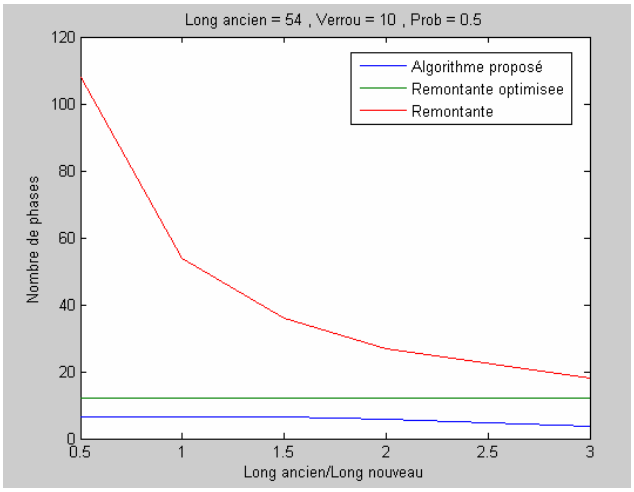

Fig. 5. Average number of reconfiguration steps versus the ratio of the new path length over the old path length.

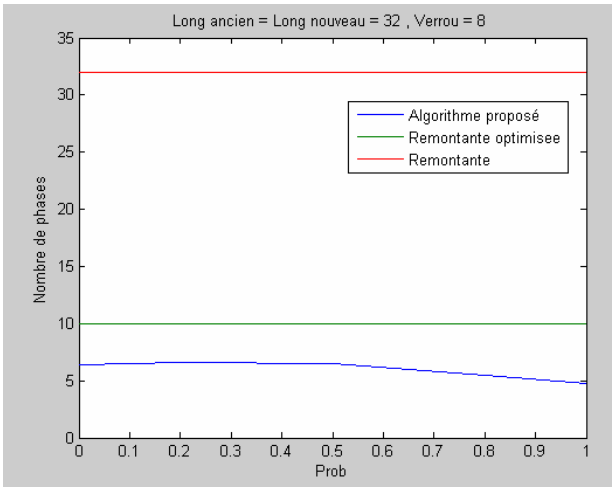

Fig. 6. Average number of reconfiguration steps versus the rate of node reutilization between old and new node.

Figure 5 shows that when we fix the length of the old path (e.g. to 54 nodes) and the number of latches (e.g. to 10 latches) as the length of the new path increases the low performance of the upstream method improves (from 108 reconfiguration steps to 18), the performance of the optimized method is constant (12 steps), and the performance of our

method slightly improves (from 6.4 steps to 3.771).

Figure 6 shows the impact of the node reutilization rate. This parameter has no influence on the upstream reconfiguration methods. The performance of our method improves for high reutilization rates.

ACKNOWLEDGMENT

We want to thank Alexandre Guitton which has participated in the initial conception of the algorithm, and Ziad Eid for his implementation of the algorithm and simulations results.

REFERENCES

[1] N. Geary, N. Parnis, A. Antonopoulos, E. Drakopoulos, J. O'Reilly, "The benefits of reconfiguration in optical networks", *10th International Telecommunication Network Strategy and Planning Symposium*, 2002.

[2] B. Jamoussi, et al., "Constraint-based LSP setup using LDP," *Internet RFC 3212*, January 2002.

[3] A. Doria, F. Hellstrand, K. Sundell, and T. Worster, "General Switch Management Protocol (GSMP) V3," *Internet RFC 3292*, June 2002.

[4] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture", Internet RFC 3031, January 2001.

[5] N. Feamster, H. Balakrihnan, J. Rexford, A. Shaikh, J. Van der Merwe, "The case for separating routing from routers", *ACM workshop SIGCOMM*, 2004.

[6] R. Mahalati, R. Dutta, "Reconfiguration of traffic grooming optical networks", *First international conference on Broadband Networks*, October 2004.

[7] Z.K.G. Patronico, P.R. Teixera, G.R. Mateus, "Traffic grooming and reconfiguration for incremental traffic in WDM optical networks", *International Network Optimization Conference*, October 2003.

[8] B. Jeager, D. Tipper, "Prioritized traffic restoration in connection oriented QoS based networks", *Computer Communications*, vol. 26, pp. 2025-2036, 2003.

[9] I. Baldine, G.N. Rouskas, "Traffic adaptative WDM networks: a study of reconfiguration issues", *Journal of Lightwave*, vol. 19, pp. 433-455, April 2001.

[10] S. Tak, P. Prathombutr, E.K. Park, "An efficient technique for a series of virtual toplogy reconfigurations in WDM optical networks", *Computer Communications*, vol. 30, pp. 1301-1314, December 2006..

[11] H. Abbu, H. Luffiyya, M.A. Bauer, "A framework for determining efficient management configurations", Computer Nettork, vol. 46, November 2004.