A Proactive Management and Rerouting Framework for QoS Critical Distributed Applications Using Active Technology

Reinaldo de B. Correia[1]
reinaldo@posgrad.nce.ufrj.br
Edmundo L.Cecilio[1]
cecilio@ime.eb.br
Luci Pirmez[1]
luci@nce.ufrj.br

Luiz F. Rust Carmo[1,2]
rustlf@utrc.utc.com
Luiz F. H. Bacellar[2]
bacellar@utrc.utc.com

[1] Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
P.O.Box 2324, 20001-970
Rio de Janeiro – RJ, Brasil

[2] United Technologies Research Center
411 Silver Lane
M/S 129-48
East Hartford, CT – 06108, USA

Abstract

In the past few years the massive deployment of real-time multimedia services has motivated the research community to investigate new quality of service (QoS) mechanisms to overcome the limitations of IP networks. Since assured levels of service must be provided, these mechanisms should interact flexibly with network performance management systems. When such integration is achieved, it is possible to trigger actions effectively to prevent QoS failures and simultaneously to balance network load. Mechanisms for rerouting traffic flows of QoS critical applications may be employed to support performance management systems in satisfying the application requirements. In addition, proactive actions could be taken before applications are affected by QoS failures. To meet these goals, a proactive network management and rerouting framework is introduced. The proposed framework is based on active technology and aims at providing the means needed for establishing new routes with sufficient resources for traffic flows of QoS critical distributed applications. A system prototype using the proposed framework was implemented and test results show that its deployment is feasible considering the hardware processing capability available today.

1. Introduction

A precise, efficient, and rapid network management system is the key mechanism to guarantee the specific quality of service (QoS) levels needed for distributed multimedia applications. In addition to being distributed and platform independent, this mechanism must have the capacity to be easily updated and yet to be flexible enough to promptly assimilate the introduction of new functionalities. Another desirable characteristic is to be proactive, taking corrective actions prior to QoS failures. In other words, the network management system should detect the QoS reduction trend and either try to reverse it, or minimize the user perception when the network fails to provide the required QoS levels. This management system should also be extended from network to workstations, communication services and applications.

Recent research has shown that Active Technology offers the flexibility and distribution assets required for management systems of the future [1]. The key concept of active technology is allowing software to be spread on demand in the network and at its border. The high degree of flexibility achieved in such a distributed processing environment allows new functionalities to be deployed to fulfill application QoS needs.

Rerouting of traffic flows through the network is being considered as an important component to improve the availability and response time of distributed systems. Since the convergence time of IP routing algorithms is usually very high [2] and is dependent on network size, rerouting real-time multimedia traffic flows is unfeasible. To address this problem, virtual circuit technologies such as the Multiprotocol Label Switching (MPLS) framework [3] are being deployed on IP networks to provide for the efficient designation, routing, forwarding, and switching of traffic flows through the network. These technologies are critical to increase the probability that application QoS needs will be satisfied by the network infrastructure.

This paper proposes a distributed architecture for network management that, besides using active technology, is totally platform independent. A proactive rerouting architecture is also introduced. It uses the proposed network management infrastructure to avoid QoS failures by redirecting application traffic flows through redundant routes.

A system based on these two proposed architectures has been implemented and used, along with a distributed multimedia system, to show the advantages and limitations of the proposed framework. The implementation also aims at: (i) investigating the

feasibility of implementing a real-time proactive performance management system running over a Java-based middleware using the mobile agent paradigm; and (ii) verifying the suitability of the proposed rerouting architecture, which is based on the same paradigm, to reroute traffic flows through less congested virtual circuits on an Multiprotocol Label Switching (MPLS) network infrastructure.

This paper is organized as follows: Section 2 presents the key concepts necessary to understand the mobility and distributed aspects of active technology for network systems; Section 3 introduces the proposed proactive management architecture; Section 4 describes the proactive rerouting architecture; Section 5 points out the tests performed over the implemented system and analyses the results and, finally, section 6 reports some of the conclusions of this paper.

2. Key concepts

The following key concepts were used as a basis for the framework proposed in this paper.

Active Technology

Active networks [4] and mobile agents [5] use computational resources inside and/or at the borders of the network to execute software on demand. The main difference between them is that active networks use (network layer) processing for packet forwarding while mobile agents execute as applications. Although the concepts of these two technologies come from different research communities to solve different problems, they overlap, and the *active technology* term has been used to specify one, another or both.

Virtual Circuits and Rerouting

Virtual circuit driven networks (ATM, Frame Relay and MPLS) stem on swapping of labels to forward packets. In particular, MPLS defines virtual circuits, known as Label Switched Paths (LSPs), over connectionless environments such as IP networks in order to support connection-oriented like services. Rerouting in virtual circuit networks is defined as the set of operations necessary for redirecting pre-established flows through redundant routes. Rerouting is generally employed to: (i) support administrative policies, (ii) to establish traffic profiles and (iii) to increase the degree of fault tolerance [6].

Traditional rerouting approaches are reactive in nature since rerouting actions are only taken after a fault is detected. In contrast, proactive rerouting must be able to find new paths before faults occur. Such proactive operations lead to significant reduction of rerouting latency. Basically, the proactive rerouting scheme consists of three set of operations: (i) identification of alternative paths: (ii) generation of local identifiers (labels) on all nodes that belong to all virtual circuits just discovered and (iii) redirection of the flow by replacing the current label.

In the context of this work, faults take place when QoS constraints are not met due to lack of resources on the current virtual circuit of an application flow. There are two approaches for redirecting flows proactively: plain and partial rerouting. Plain rerouting aims at replacing the whole virtual circuit of a pre-established flow with a new redundant virtual circuit, while the partial one only replaces a section of the current virtual circuit.

The partial approach presumes there is a section that must be identified in the current virtual circuit. This section, the critical section, is the one, which is the bottleneck of the virtual circuit considering the QoS metric relevant to the flow (e.g., lowest bandwidth, higher delay). Preliminary results establish an exponential relationship between path length in hops and path computation time, justifying the adoption of partial rerouting [7]. The idea behind partial rerouting is to restrain the searching area for alternative paths so that the number of nodes and links involved in rerouting operations becomes small. The final result is a lower processing time and consequently, lower rerouting latency. Moreover resource consumption in rerouting tasks is also reduced turning the scheme more scalable.

3. Proactive Management Architecture

In an active technology environment, manageable elements (workstations and the network devices) will have no more than basic downloading capabilities, allowing them to pack specific functionalities on demand. These on demand applications can be used to enable advanced and local processing of management information, reducing substantially the reaction time and the network management extra traffic (traditional polling of management information can be practically eliminated).

One critical disadvantage of centralized management approaches (e.g., SNMP) is the difficulty to get precise timing measures [8]. Sending a mobile agent, or an active application, permit to effectuate and evaluate the measures locally (reacting, whenever is necessary) without needs of generating extra network traffic.

Additionally, local applications can calculate variations on critical parameters and detect their behavior tendency. Actions can be executed locally or notifications can be sent to management servers when appropriate limits of these tendencies were disrespected. With these tendencies detected, it is possible not only to estimate when minimum limits of QoS will be reached but also, depending on the amount of time remaining, to take actions to try to revert the malicious trend or, at least, to try to minimize the consequences of QoS failures, if they occur. By this way, a proactive performance management can be implemented. A proactive management can be

realized only if it is possible to execute automatic actions. Active technology is a good way to get this.

In spite of the flexibility offered by active technology, efficiency and efficacy of the real time monitoring and automatic execution of actions can be questioned. Active technology has to use a mobility infrastructure and this will introduce more layers of processing. The use of a platform independent language, as Java, will further increase another layer. Test results show that this management scenario is possible using hardware that is not so powerful.

The work described in this paper is based on the AGAD (Distributed Management Architecture using Active Technology) developed at the Federal University of Rio de Janeiro (UFRJ), Brazil. AGAD was extended to provide proactive performance management, and validated by means of one of its potential applications: rerouting of flows.

Figure 1 presents AGAD architecture, which provides necessary infrastructure for a distributed management of workstations, their applications, services and network elements. Master Manager (MM) is a fixed application and is responsible for managing a set of domains, each of them managed by a Domain Manager (DM). DMs are created, sent to their domains and monitored by MM. A domain is a set of network elements, workstations, applications and services supervised by the same administrative authority. DM creates and sends Inspectors to the elements to be managed. In the case explored in this paper, Inspectors collect QoS parameter data locally, analyze them as required and start actions or send notifications to the DM. Inspectors use whatever is available locally, as MIBs or interfacing directly with operating systems to get necessary data. Inspectors were developed as dynamically configurable, being updated during their execution, if it is required.

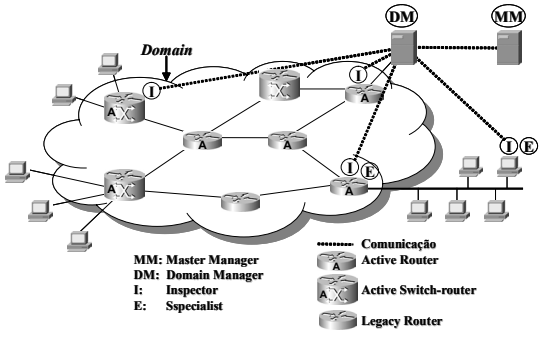

Figure 1: Overview of AGAD

When necessary, DM, after analyzing notifications received from Inspectors, sends Specialists to the elements where they are necessary to start specific actions to maintain QoS parameters in their required levels. Examples of these specific actions are changing of scheduling policies, reconfiguring buffers utilization, interacting with an adaptive application, changing a protocol or simply delivering a message to a rerouting system.

As mentioned before, this paper focuses on the performance management capabilities (PM) provided by AGAD. PM has two orthogonal objectives: (i) to provide appropriated QoS levels for users, and (ii) to enable allocation of resources, which maximizes the utilization of the managed workstations and network elements. A Performance Management System, as organized by Pacifici [9] and showed in Figure 2, acts only based on control parameters. These are monitored and, when necessary, set. There are no interaction between PM and Real Time Traffic Control, which is a set of mechanisms responsible for controlling specific functionalities, like scheduling, rerouting, allocation of memory, protocols, admission control, among others.

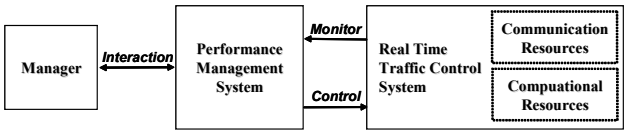

Figure 2: Pacifici Model

During performance management, Inspectors collect data from managed elements and analyze the variation of the monitored parameters. These could be from applications, operating system, communication services, protocols or devices. To detect QoS failures tendencies, variations are compared with pre-configured limits. In case of disrespect of these pre-configured limits, Tendency Alarms are sent to Domain Manager, which analyzes these alarms and extrapolates theses variations in order to define which actions are to be started and in which elements these operations will be carried out. In this way, it is possible to revert the tendency or to minimize their consequences.

A proactive performance management can be used to try to prevent or to minimize consequences of QoS failures in many situations. Adaptive applications can be notified to change media formats without compromising quality of presentation. By this way, QoS requirements can be reduced.

4. Proactive Rerouting Architecture

The rerouting functionalities were decomposed into three agents to make their code size smaller and, therefore, reduce the installation and migration latency requirements.

The IngressNodeAgent (INA) once installed in the ingress node of the virtual circuit starts its rerouting management tasks. It also has the responsibility to create the two other agents and to handle external messages (e.g., from the Proactive Management System). The PathNodeAgent (PNA) identifies the intermediary nodes

of the virtual circuit to install its replicas. Upon reaching the target nodes, PNA replicas start to monitor the local intermediary node states about relevant QoS parameters.

The SearchRouteAgent (SRA) is responsible for searching new paths around the critical section of the virtual circuit. SRA must also be able to redirect the flow just when it receives a rerouting request indicating a trend to a QoS failure.

The relationship among the rerouting agents, as well as between the Rerouting Architecture and an external system, is covered by five distinct phases. Figure 3 shows graphically these relationships.

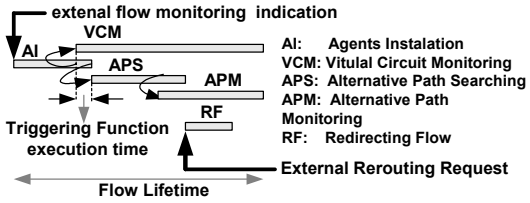

Figure 3: Rerouting Phases

The operations executed by these components are triggered by two external events: (i) beginning of an application flow monitoring and (ii) detection of a trend to a QoS fault.

Once INA agent captures an external event indicating that a flow is being monitored, all phases, except RF phase, are executed sequentially. Theses phases are considered proactive because their operations are started, or finished, before external rerouting request event is generated, signaling a QoS Failure or a tendency of QoS fault. RF phase is reactive since its operations are started afterwards. By executing most of the phases in advance, it is possible to lessen the rerouting latency.

Agents Installation (AI) and Virtual Circuit Monitoring (VCM) Phases

Figure 4 describes all main operations involved in these phases. This figure shows, as an example, a 5-hop-length virtual circuit in which the critical section is formed by nodes C, D and E. The dotted arrows refer to the VCM phase operations.

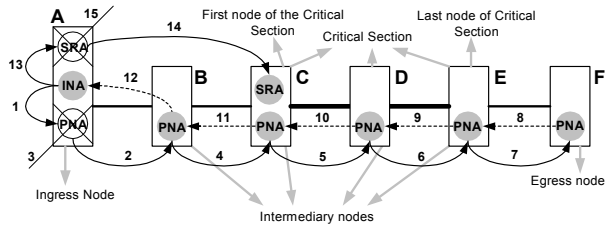

Figure 4: AI and VCM Phases

As soon as INA agent starts, it creates PNA agent (operation 1) which travels (operation 2) to the downstream node (node B) in the virtual circuit. On node B, PNA agent clones itself. The PNA clone identifies the next hop and travels to it (operation 4). These operations (operations 5, 6 and 7) are repeated until one of the PNA copies reaches the egress node (node F), initiating the VCM phase. The PNA clone hosted in the egress node (node F) sends periodically the local QoS parameters to the PNA clone in the upstream node (operation 8). On the upstream node (node E), this QoS information is aggregated to the local one and sent to the next upstream node. In this way, during the VCM phase, the QoS information of all nodes is delivered to INA agent (operations 9 – 12). INA agent uses this QoS information to figure out the critical section (nodes C, D and E) of the virtual circuit. At this time, INA agent while creating SRA agent, hands over to it the address of the first node of the critical section (operation 13). SRA agent just travels to the first node of the critical section (node C) and begins executing to finish the AI phase.

INA agent also implements a triggering function to diminish the side effects produced by the position changing of the critical section. Due to traffic patterns, as well as load unbalance on network nodes and links, it is expected that the position of the critical section changes over time. The triggering function only allows to create a new SRA agent whenever three consecutives critical section calculations lead to the same results.

Alternative Path Searching (APS) and Alternative Path Monitoring (APM) Phases

APS phase starts when SRA agent begins to execute its first instruction on the first node of the critical section, and ends as soon as the last replica reaches the last node of the critical section, pointing out that the last alternative path was discovered. The description of this phase is graphically presented in Figure 5. This figure is similar to Figure 4, including two alternative paths in the searching area around the critical section. The dotted arrows represent the alternative path update messages.

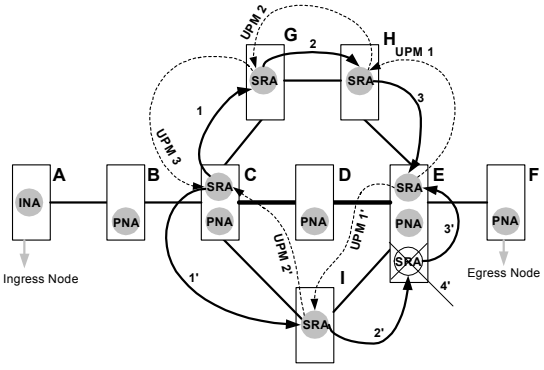

Figure 5: APS and APM phases

SRA agent interacts with the local operating system (node C) to obtain all active interfaces through which

SRA clones are launched next. These operations are carried out until SRA agent copies arrive at the last node of the critical section (node E). The adopted scheme for searching alternative paths around the critical section was flooding. To limit the searching area, which encompasses the nodes and links involved in searching tasks, a specific private variable was incorporated in the SRA agent code. This variable specifies the maximum alternative path length. For every hop, this variable is decremented and if a zero value is reached, the SRA stops running. Hence, the flooding of SRA agent is restricted to the searching area.

The first SRA agent replica to host in the last node of the critical section begins to send update messages (UPM 1) to the upstream alternative path node. After that, as new replicas arrive through the other interfaces, they transfer the incoming interface (operation 3') identifier to the first replica and stop running (operation 4') immediately. Then, update messages will also be sent through this new interface (UPM 1').

APM phase consists of sending messages with the local QoS information of the alternative paths to the SRA agent hosted in the first node of the critical section (node C). These operations are similar to VCM phase. The SRA agent on node C builds an information base to choose the best alternative path on receipt of a rerouting message.

Redirecting Flow (RF) Phase

In RF phase, operations are triggered after an external rerouting request reaches the INA agent in the ingress node. This is the reason why this phase is said reactive. The time constraint is an issue because high latencies in this phase can jeopardize application performance. Nevertheless, another concern that is tightly coupled with RF phase latency is the label assignment on the alternative path nodes. If labels are assigned during the proactive APM phase, RF phase latency is reduced at the cost of a higher number of labels consumed. That is, in APM phase, labels on all nodes in the searching area must be assigned in advance. Depending on the number of nodes, the degree of redundancy of the searching area, number of flows being rerouted and number of virtual circuits, the label spaces of searching area nodes may be exhausted. Hence, RF phase can adopt two possible approaches: the early or on-demand label assignment.

Figure 6 shows RF phase operations for both approaches. SRA agent located at the first node of the critical section (node C) chooses the best alternative path and redirects the flow by changing the labels (operation 6) after having received a rerouting message from INA agent (operation 1). That is, on early assignment approach, operations 2 through 5 are not executed. After operation 2 is finished, SRA jumps directly to operation 6. Operations 2 through 5 are needed for the on-demand label assignment approach.

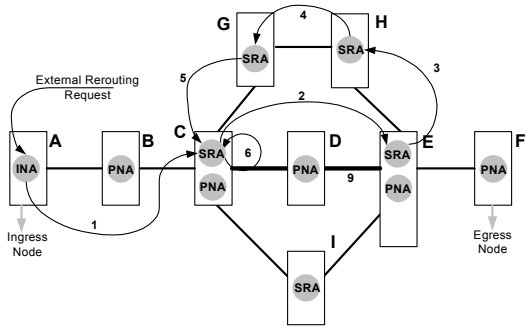

Figure 6: RF Phase Early and on-demand approach

These operations are related to the creation and assignment of labels on the alternative path that was selected previously (operation 2). It is worth noting that while on the former approach the labels were assigned for all alternative paths discovered in the searching area, on the latter, labels were only consumed for the nodes of the chosen alternative path.

5. Implementation and Tests

The platforms for development and testing of the prototypes were PC based hardware with Linux as the underling OS. Java (JDK- version 1.3.1) was the language adopted for coding the components - agents - of both architectures. A mobile infrastructure called μCode [10] was incorporated to the Java virtual machines of the test platforms. This infrastructure provides a computing environment for execution of mobile threads. According to Fuggeta's taxonomy [11], μcode furnishes weak mobility because agent data state remains unaltered after migration while the execution state is lost.

Perl scripts were written to provide the means for the interaction between the rerouting agents and OSes. IP addresses and labels are information that must be available to the rerouting agents as well as the support for reconfiguring routing and forwarding tables on the fly. The rerouting agents were tailored for redirecting flows with bandwidth and latency constraints on a MPLS-IP network infrastructure.

The management and rerouting tests aim at verifying whether the proposed proactive schemes working together in the same framework are able to avoid QoS failures. Besides, it will be possible to conclude the feasibility of Java and μCode as the underling mobile environment.

5.1 Proactive Management Tests

In the first test, it was measured how much time was spent for the Specialist to get ready to start some action in a managed element. Figure 7 presents an overview of the test.

Inspector gets measures from the managed element and detects a tendency of QoS failure in a parameter. Then Inspector prepares and sends, via Java socket a

Tendency Alarm to its Domain Manager. DM analyses this alarm and decides which Specialist to send to which managed element (in this case, the same one of the Inspector). The Specialist is sent by µCode only if the Specialist code is not present in the class space of the destination element otherwise a reference is forwarded. After that, the Specialist is instantiated and put to run. All processing phases are represented in Figure 8.

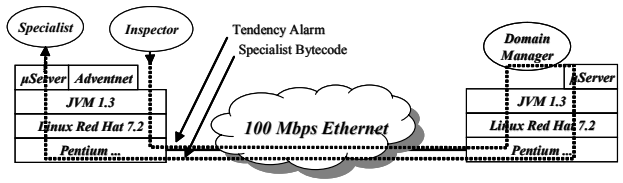

Figure 7: Overview of the first test

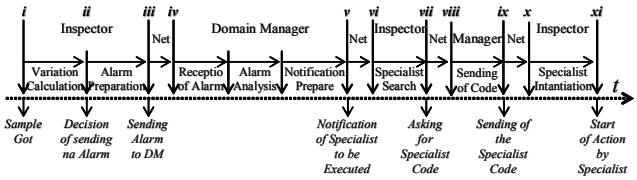

Figure 8: Phases of the test

There could be as much as four network transits (alarm sending, Specialist type notification, Specialist code request and Specialist code shipment) and all layers of processing showed in Figure 7. The alarm analysis in DM was not accomplished in this test. This analysis may require reasonable complexity computations, once techniques of artificial intelligence and/or access to a database with the baseline of the network may be needed. Although, DM is unique for an entire domain, as a corporation network, for instance, it could be executed in a workstation with enough processing capacity. The results are presented in Table 1.

Table 1: Results of the first test

	Subtest 1	Subtest 2	Subtest 3
Mean:	136.80	42.23	433.40
Adjust to Normal (%):	99.99	95.48	95.30
Standard Deviation:	7.24	5.39	6.27
Conf interval of 95%:	±2.59	±1.93	±2.24

In subtest 1, the Specialist code size was 1 KB and systems were rebooted to eliminate the presence of code in cache. That is, at all times, the transmission of the code was required. In subtest 2, systems were not rebooted, so codes were in cache and not retransmitted. In subtest 3, systems were rebooted and Specialist had 10 KB of byte codes.

The worst case – Specialist with 10 KB without cache – required 433 ms to start an action at the managed element. If network delay was 120 ms, a very large delay for a intra domain network, 480 ms have to be added, totalizing 913 ms. These numbers indicate that 1 second is a reasonable limit between detection of a Tendency Alarm and starting an action.

The second test consists of monitoring delays in a multimedia application with a server and a client streaming a video with 400 Kbps. The network used (shown in Figure 9) is from a university school with twelve different departments and is a typical network where multimedia applications will be commonly used in the future.

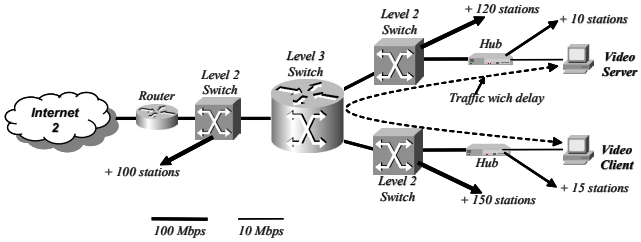

Figure 9: Network infrastructure for test 2

Delay was monitored at 0,2 Hz and the limits set to generate a Tendency Alarm by the specialist was two: (i) when accumulated delay, not considering the number of samples, increase 40 ms, and (ii) when, in one interval between two samples, delay increase 35 ms. It was considered that a delay of 90 ms causes a QoS failure. The measures of delay were repeated for weeks and the most significant two-hour (14 to 16h) period for our analysis was selected. This is shown in Figure 10. During other periods the traffic of the network was extremely light, being difficult to get QoS failures. The objective of the test is to measure the efficacy of detecting QoS failure tendency. Results are summarized in Table 2.

Table 2: Results of test 2

Fact	Qtty	
Two hour samples:	720	
QoS failures:	49	100 % (Reference)
QoS failures preceded of alarms:	24	49.0 %
Alarms generated 5s before failure:	10	20.4 %
Alarms generated 10s before failure:	7	14.3 %
Alarms generated 15s before failure:	5	10.2 %
Alarms generated 10s before failure:	2	4.0 %
Alarms generated at the failure:	15	30.6 %
Failures not preceded by alarms:	8	16.3 %

Among 720 samples, 49 presented latencies above 90 ms.. A sequence of adjacent samples with delay 90 ms or superior is considered as 1 time, once the increase of the

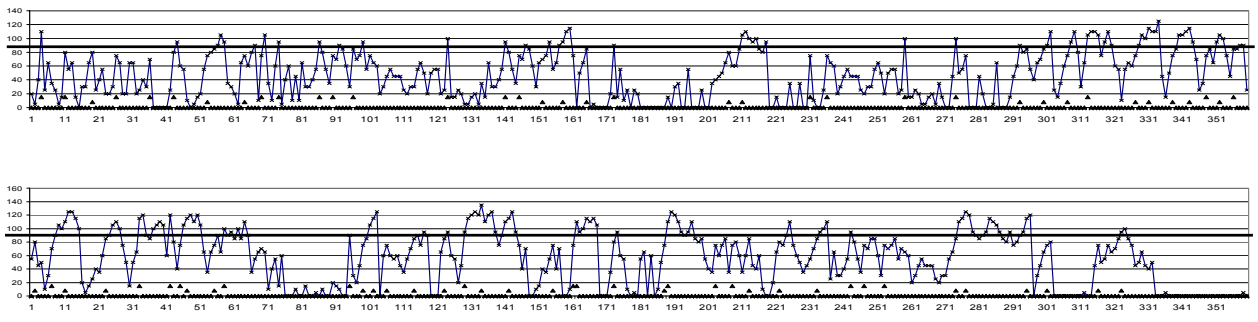

Figure 10: 720 Delay samples and alarms for 2 hours (14-15h above, 15-16h below)

variation causes QoS failure and not the maintaining of a high delay without abrupt variation. Of these 49 failures, 24 of them were preceded of Tendency Alarms within 5 to 20 seconds before they occur. That is, 49% of efficacy. 15 of the failures were detected only at the sample they occur, that is, without any anticipation. These are inutile alarms. In 8 failures there wasn't any alarm generated. This happened because delay was high at the time, varying little, but above and below 90 ms. 24 of the Tendency Alarms generated were false. They were generated but no QoS failure occurred within 20 s after the alarm.

In this test there were no actions started by specialists. These actions could have prevented some of the QoS failures

5.2 Proactive Rerouting Tests

A prototype was implemented to verify if the proposed proactive rerouting architecture is able to reroute pre-established flows before the occurrence of a QoS fault. In the context of this article, the CR phase latency must be confronted with the time that the proactive management detects tendencies of QoS faults in advance. The total latency of the proactive rerouting phases, except CR one, is also important because it establishes the minimum lifetime of flows, which can be rerouted. In other words, flows can be rerouted only if their lifetimes are higher than the total latency of all these phases. This latency is the time elapsed between the installation of INA agent in the ingress node and SRA agent receipt of the first update massage, indicating that the first alternative path was found out.

Table 3: Test Results – Proactive Phases (ms)

Values	AI phase			VCM phase	APS phase	APM phase	
	1	2	3			**1**	2
Avg	821	580	321	33.5	326	27.9	74.7
STD	41	20	4.58	0.35	15	0.71	3.89
Conf 95%	11.4	5.4	1.3	0.1	4.2	0.2	1.1

To measure phase latencies, five tests were established. These tests correspond to the rerouting phases identified previously. As phase latencies depend on virtual circuit (VC) and alternative path (AP) lengths, each test was also divided into subtests. By measuring subtest latencies, it is possible to derive the relationship between the total rerouting latency and the virtual circuit length as well as the alternative path length.

Tables 3 and 4 summarize the test results presenting the values measured. K6-II/500Mhz machines with Linux Red Hat 7.2 and a patched Kernel version 7.4.19 were used as MPLS routers.

Table 4: Test Results – Reactive Phase (ms)

Values	RF phase				
	RF 1	**RF 2**		**RF 3**	
		1	3	**A**	**C**
Avg	4	4	90	4.5	122
STD	0	0	2.87	0.07	3.95
Conf 95%	0	0	0.8	0.02	1.09

The subtests underlined in Table 3 are those that should be executed as many times as the number of virtual circuit hops while the bolded ones are repeated a number of times equal to the number or alternative path hops.

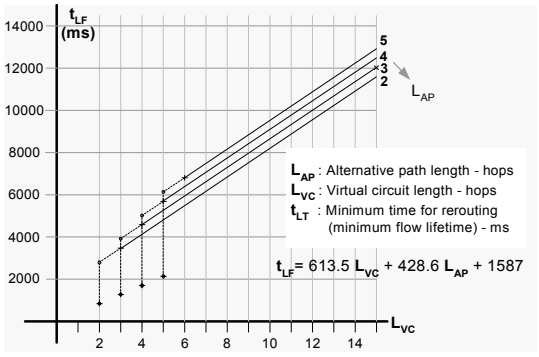

Figure 11: Min time for rerouting - Early approach

Computing the values in Table 3 in accordance with the length dependency nature of each subtest, the derivation of the expressions and graphics in Figures 11 and 12 are straightforward. Figures 11 and 12 present the results considering the early and on-demand label assignment approaches respectively.

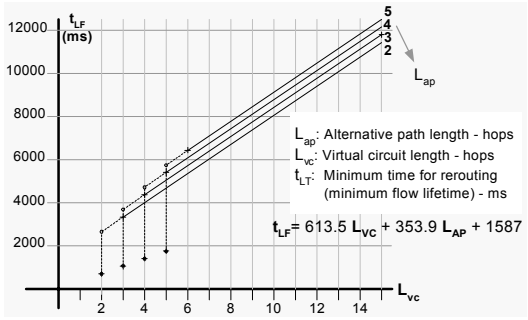

Figure 12: Min time rerouting - On-demand approach

To make clear the lifetime flow limitation, Figure 11 shows that it is only possible to reroute flows with lifetime longer than 11.9 s for alternative paths lengths shorter than 4 hops and virtual circuit length equals to 14 hops. Theses graphics also show that the minimum time for rerouting (minimum flow lifetime) is lower when on-demand label assignment scheme is adopted. In this case, the RF phase latency increases.

RF phase latency can also be calculated with the values in Table 4. For on-demand label assignment approach, this latency is equal to the sum of tests 1 and 2 latency values, 98 ms.. Early label assignment scheme makes RF phase latency be dependent on alternative path length due to the additional operations needed (as shown in Figure 13).

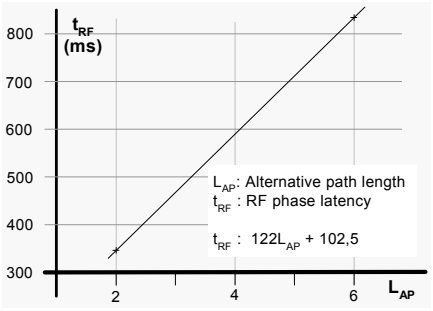

Figure 13: RF phase latency

To reroute a flow (RF phase), it is necessary 834.5 ms for a 5-hop-length alternative path. The virtual circuit length does not impact the rerouting time if the partial proactive rerouting approach is adopted. Hence, as the searching area size limits the alternative path length, it could be adjusted accordingly. The searching area size is a trade-off between the probability of finding alternative paths and the rerouting latency. Finally, the results show that rerouting flows proactively in a partial manner is feasible since all latency values were far below the time the proactive management prototype is able to detect tendencies of QoS failures.

6. Conclusions

This paper introduced proactive management and rerouting architectures, which work together, establishing a framework to enhance the underling network functionalities for furnishing QoS to critical applications such as real-time multimedia.

Besides the presence of many software layers, real time proactive performance management is already possible. Nevertheless, further studies are necessary to establish the limits of the proposed scheme.

The proactive rerouting architecture was presented as a tool for management systems to redirect critical application flows through alternatives paths in a trial to revert the QoS fault tendency. The partial rerouting was highlighted and its benefits as well as its shortcomings described. Test results of the prototypes and the confrontation of both demonstrated that it worth devoting more effort in investigating theses approaches for delivering QoS. The conception of the proposed framework based strongly on the proactive principle reveals the authors' belief that the apparent randomness of network traffic is tractable. In addition, this randomness may be managed on network design and management levels in such a degree that, in the future, this framework could help providing satisfactory QoS over a connectionless environment.

References

[1] Kawamura, R. and Stadler, R., "Active Distributed Management for IP Networks", IEEE Communications Magazine, 2000.

[2] Labovitz C. et al, "Delayed internet routing convergence", In Proc. ACM SIGCOMM '00 pp. 175–187, Stockholm, Sweden, 2000.

[3] Rosen E. et al, "MPLS Architecture", RFC–3031, IETF January 2001.

[4] Tennenhouse, D.L. et al, "A Survey of Active Network Research", IEEE Comm.s Mag., Jan 1997.

[5] Bradshaw, J. et al, "Software Agents", The MIT Press, Menlo, California, 1997.

[6] Autenrieth, A. and Kirstdter, A. "Fault-Tolerance and Resilience Issues in IP-Based Networks, Second International Workshop on the Design of Reliable Communication Networks (DRCN), April 1995.

[7] G. Apostolopoulos et al, "Intradomain QoS Routing in IP Networks: A Feasibility and Cost/Benefit Analysis", IEEE Network Magazine, 1999.

[8] Bieszczad, A., Pagurek, B. and White, T., "Mobile Agents for Network Management", IEEE Communication Surveys, Fourth Quarter, 1999.

[9] Pacifici, G. and Stadler, R., "An Architecture for Performance Management of Multimedia Networks", IFIP/IEEE International Symposium on Integrated Network Management, 1995.

[10] Picco, G.P., "μCode: A Lightweight and Flexible Mobile Code Toolkit", Proceedings of the 2nd International Workshop on Mobile Agents 98, 1998.

[11] Fuggeta, A. et al, "Understanding Code Mobility", IEEE Transactions on Software Engineering, IEEE, 1998.