ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE
CONCENTRATION RÉSEAUX DE TÉLÉCOMMUNICATION
M.Ing.

PAR
Bachir TOUTI

CONCEPTION ET IMPLÉMENTATION D'UN GESTIONNAIRE DE FLUX SIP

MONTRÉAL, LE 17 AOÛT 2011

PRÉSENTATION DU JURY

CE RAPPORT DE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Michel Kadoch, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Nicolas Constantin, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Stéphane Coulombe, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 2 AOÛT 2011

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

CONCEPTION ET IMPLÉMENTATION D'UN GESTIONNAIRE DE FLUX SIP

Bachir TOUTI

RÉSUMÉ

Le protocole SIP est largement utilisé par l'industrie des télécommunications. Cela implique que le nombre d'utilisateurs du service relié est très élevé. Ainsi, la QoS exige que les composants du réseau SIP restent fonctionnels en tout temps. Toutefois, les rafales de trafic qui arrivent d'une manière inopinée causent beaucoup de problèmes de performance, car les composants principaux du réseau tels que les serveurs SIP peuvent être surchargés. La retransmission déclenchée par UDP, habituellement, empire la situation. En conséquence, plus d'appels sont rejetés et les délais augmentent considérablement.

Les sessions SIP sont basées sur l'échange de messages. Chaque message a une tolérance différente au délai et au rejet. Le traitement des messages devrait être effectué dépendamment du type du message, à travers la classification et la priorité des messages. Pour plus de performance, le réseau SIP devrait se disposer des mécanismes capables de gérer les messages SIP convenablement.

Une nouvelle approche de gestion des messages SIP a été introduite afin d'améliorer la performance du réseau SIP en cas de rafales de trafic. Un traitement différencié des messages SIP a été implémenté dans un GFSIP afin de traiter les demandes et les réponses SIP différemment. L'approche se base sur différentes tolérances dans la même session SIP. Trois disciplines de file d'attente ont été élaborées pour étudier leurs effets sur le délai et le rejet. L'étude des disciplines dans le cas d'un réseau SIP était intéressante, car la signalisation SIP est organisée sous forme de session.

Le diagnostic de performance SIP a été réalisé grâce à un banc d'essai SIP. L'échange de messages entre les entités SIP a été illustré par un scénario proposé. Les modèles de Markov ont été utilisés pour illustrer les étapes de signalisation et leur relation avec la tolérance. Durant les tests, les rafales ciblaient le GFSIP et le serveur SIP a été protégé. L'analyse a été basée sur la variation des délais et des taux de rejet. Les résultats ont conclu l'effet des différentes disciplines sur la QoS.

Mots clés: SIP, QoS, et GFSIP

CONCEPTION ET IMPLÉMENTATION D'UN GESTIONNAIRE DE FLUX SIP

Bachir TOUTI

ABSTRACT

SIP is widely used by the telecom industry. This implies that the number of users using related services is very high. Thus, QoS requires that SIP network components remain functional at all times. However, traffic burst usually harms the network performance because main components such as SIP server could be overloaded. UDP retransmissions usually exacerbate the situation. As a result, more calls are rejected and signaling delays increase drastically.

SIP sessions are established based on message exchange. Each message has a different tolerance to delay and rejection. Message processing should depend on the message type through scheduling and message priority. SIP network should possess mechanisms capable of managing SIP message conveniently.

A new approach of SIP message management has been introduced in order to improve SIP network performance in case of traffic burst. A differentiated handling of SIP messages has been implemented in a GFSIP in order to process SIP requests and responses differently. Different tolerances have been established for the same SIP session. Three queuing disciplines have been implemented to study their effect on delay and call rejection. The study of queuing discipline in this context was challenging because SIP signaling is organized into sessions.

The diagnosis of SIP performance was carried out through a SIP test-bed. Message exchange between SIP entities has been illustrated via a proposed scenario. Markov Models have been used to illustrate signaling steps and their relationship with the tolerance. During tests, bursts targeted the GFSIP and the SIP server was protected. Analyses were conducted based on delays and rejection rates variation. The result demonstrated the effect of different queuing disciplines on QoS.

Keywords: SIP, QoS, and GFSIP

TABLE DES MATIÈRES

Page

INTRODUCTION 1

CHAPITRE 1 SESSION INITIATION PROTOCOL 7
1.1 Les éléments d'un réseau SIP.....8
1.1.1 Les UAs (User Agent) 8
1.1.2 Les serveurs SIP..... 8
1.1.3 Les serveurs de localisation (Location Server)..... 9
1.2 Les messages SIP 9
1.3 L'adressage.....11
1.4 Le dialogue.....11
1.5 La congestion dans les Réseaux SIP 15
1.5.1 Les causes de la congestion dans les réseaux SIP..... 16
1.5.2 Le contrôle de la congestion dans les spécifications SIP..... 16
1.5.3 Les travaux de recherche antérieurs..... 17
1.6 Conclusion 21

CHAPITRE 2 L'ÉTAT DE L'ART 23
2.1 La tolérance aux délais des messages SIP.....24
2.2 Les rejets et la priorité des messages SIP.....24
2.3 La gestion des messages SIP et l'approche proposée 27
2.4 Les files d'attente et les messages SIP 29
2.5 Conclusion.....29

CHAPITRE 3 L'ARCHITECTURE EXPÉRIMENTALE DE TEST 31
3.1 Topologie : Serveur - Serveur vs UA – Serveur 31
3.2 L'architecture standard de test et Modules utilisés 33
3.2.1 Le modèle utilisateur..... 34
3.2.2 Le modèle enregistrement..... 35
3.2.3 Le modèle UAC 35
3.2.4 Le modèle UAS..... 37
3.2.5 Les délais, les rejets et les modèles utilisés 38
3.3 L'architecture de test proposée 39
3.4 L'architecture du gestionnaire du flux SIP proposée 41
3.5 Conclusion.....44

CHAPITRE 4 SCÉNARIOS, IMPLÉMENTATIONS ET TECHNOLOGIES UTILISÉES.....45
4.1 Les Servlets SIP de Java.....45
4.2 Bref aperçu sur SIPp 46
4.3 Les scénarios utilisés.....48
4.3.1 Le flux de paquet de signalisation et les délais associés..... 49

4.3.2 Le flux de paquet d'enregistrement 52
4.4 Phase de l'implémentation 53
4.4.1 L'implémentation des UAs 53
4.4.1.1 L'instance SIPp Client-Appel 54
4.4.1.2 L'instance SIPp Client-Enregistrement 56
4.4.1.3 L'instance SIPp serveur 58
4.4.2 L'implémentation du Serveur SIP 59
4.4.2.1 Le proxy 60
4.4.2.2 L'enregistrement 63
4.4.3 L'implémentation du GFSIP proposé 64
4.5 L'architecture de l'environnement expérimental 67
4.6 Le déploiement 69
4.7 Limitation 71
4.8 Conclusion 71

CHAPITRE 5 TESTS ET RÉSULTATS 73
5.1 La dépendance entre messages SIP 73
5.2 Tests et statistiques 75
5.3 L'effet du ramasse-miettes (Garbadge Collector) 76
5.4 Le système FIFO-GFSIP 80
5.4.1 L'analyse de la langueur des files d'attente 82
5.4.2 L'analyse des délais et rejets 84
5.4.3 L'effet de la longueur maximale sur la tolérance 88
5.5 Le système FQ-GFSIP 89
5.5.1 L'analyse des résultats 90
5.5.2 Les rejets 95
5.6 Le système PQ-GFSIP 96
5.6.1 La priorité des messages SIP 97
5.6.2 L'analyse des résultats 100
5.7 FIFO-GFSIP vs FQ-GFSIP vs PQ-GFSIP 104
5.7.1 Les délais 104
5.7.2 Les rejets 106
5.7.3 Conclusion 107

CONCLUSION ET TRAVAIL FUTUR 109

RECOMMANDATIONS 111

ANNEXE I LE CODE DES MÉTHODES IMPLÉMENTÉES DANS LE SERVEUR SIP 113

ANNEXE II LECODE DE LA CLASSE UTILISÉE POUR LES FILES D'ATTENTE 117

ANNEXE III LE CODE DU CLASSIFICATEUR 121

ANNEXE IV LE CODE DE L'ORDONNANCEUR AVEC LA DISCIPLINE FQ 123

ANNEXE V LE CODE DE L'ORDONNANCEUR AVEC LA DISCIPLINE PQ 127

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES.....131

LISTE DES FIGURES

Page

Figure 1.1 Le message 'Invite' 10

Figure 1.2 La signalisation SIP dans un appel VOIP 12

Figure 1.3 Le scénario de la messagerie instantanée 13

Figure 1.4 Les scénarios de Présence 14

Figure 2.1 Les types de rejet 25

Figure 3.1 La topologie Serveur-Serveur 31

Figure 3.2 La topologie UA-serveur 32

Figure 3.3 Un système standard d'évaluation SIP 33

Figure 3.4 Le modèle utilisateur de Markov 34

Figure 3.5 Le modèle Enregistrement de Markov 35

Figure 3.6 Le modèle UAC de Markov 36

Figure 3.7 Le modèle UAS de Markov 37

Figure 3.8 L'architecture de test proposée 40

Figure 3.9 L'architecture du GFSIP proposé 42

Figure 4.1 Le scenario proposé d'un appel VOIP 50

Figure 4.2 Le processus d'enregistrement 53

Figure 4.3 SipSession vs SipApplicationSession 61

Figure 4.4 L'algorithme implémenté dans le GFSIP 65

Figure 4.5 L'architecture de l'environnement expérimental 68

Figure 5.1 Le diagramme de dépendance entre messages SIP 74

Figure 5.2 L'effet des algorithmes du GC sur le délai 77

Figure 5.3 L'effet de l'YG sur les délais 79
Figure 5.4 L'effet de la longueur des files d'attente sur le délai et le rejet 81
Figure 5.5 L'évolution de la longueur des files d'attente 82
Figure 5.6 Le délai 'Invite-180' 84
Figure 5.7 Le délai '200-Ack' 84
Figure 5.8 Délai 'Invite-180' vs délai '200-Ack' 86
Figure 5.9 Le nombre d'appels rejetés 87
Figure 5.10 Le délai 'Invite-180' 90
Figure 5.11 Le délai '200-Ack' 90
Figure 5.12 Le délai 'Invite-180' vs délai '200-Ack' 91
Figure 5.13 L'évolution de la taille des files d'attente 92
Figure 5.14 L'effet du TS sur le délai '200-Ack' 94
Figure 5.15 Le nombre d'appels rejeté 96
Figure 5.16 Le mécanisme de gestion des priorités 97
Figure 5.17 L'algorithme de priorité implémenté 100
Figure 5.18 Le délai '180-Invite' 101
Figure 5.19 Le délai '200-Ack' 101
Figure 5.20 Le délai 'Invite-180' vs '200-OK' 102
Figure 5.21 Le nombre d'appels rejeté 103
Figure 5.22 L'effet des disciplines sur le délai 'Invite-180' 104
Figure 5.23 L'effet des disciplines sur le délai '200-Ack' 105
Figure 5.24 L'effet des disciplines sur le rejet 107

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AAA	Authentication Authorization Accounting
Ack	Acknowledge
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
B2BUA	Back-to-Back User Agent
CCT	Control Through Thresholds
DNS	Domain Name System
DoS	Denial of Service
FIFO	First In First Out
FQ	Fair Queuing
GC	Garbage Collector
GCS	Glassfish Communication Server
GFSIP	Gestionnaire de Flux SIP
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSR	Java Specification Requests
MI	Messagerie Instantanée
PCAP	Packet Capture
PQ	Priority Queuing
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RFC	Request For Comment
RTP	Real Time Protocol
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SSL	Secure Socket Layer

TCP	Transmission Control Protocol
TLS	Transport Layer Security
TS	Temps de sonneries
UAC	User Agent Client
UAS	User Agent Server
URI	Universal Resource Identifier
UDP	User Datagram Protocol
VOIP	Voice Over IP
XML	Extensible Markup Language
YG	Young Generation

INTRODUCTION

SIP (Signaling Initiation Protocol) (J.Rosenberg, 2002) est un protocole de signalisation conçu pour établir et gérer les sessions entre deux entités communicantes dites UAC (User Agent Client) et UAS (User Agent Server). Il se caractérise par sa facilité ainsi que sa flexibilité d'intégration dans les architectures multimédia. Il a été utilisé dans plusieurs types d'application telle que VOIP, présence et la messagerie instantanée. Sa performance lui a permis d'être adopté comme le protocole de signalisation pour les réseaux de nouvelles générations.

Beaucoup de recherches ont été effectuées sur l'utilisation du protocole SIP dans le but d'aboutir à un système complet et fiable. Il existe, toutefois, certains problèmes de surcharge des serveurs. Cette surcharge est essentiellement causée par la nature de trafic voix qui arrive en rafale. Le système se retrouve donc dans une baisse de performance.

Plusieurs raisons peuvent engendrer une rafale de trafic : les périodes occupées pendant la journée dans le cas d'un centre d'appel, les attaques de dénis de service (DoS) ou encore des demandes d'enregistrement (Register) après le rétablissement du courant électrique. Le trafic arrive d'une manière brusque et avec un volume, souvent, au-delà de la capacité du serveur, ce qui mène à une dégradation de ses performances.

Un système robuste devrait avoir les mécanismes nécessaires afin d'être immunisé contre tout type de problème, notamment si sa popularité accroît et son utilisation augmente plus rapidement. Le protocole SIP est considéré, toujours, comme un bon candidat pour la signalisation et la gestion des sessions des applications de nouvelles générations. On s'attend à un déploiement omniprésent dans toutes les infrastructures de télécommunication, ainsi qu'une augmentation dramatique dans la charge du trafic, ce qui constitue principalement la motivation de base de tous les travaux de recherche qui ont ciblé ce protocole.

Un serveur SIP est congestionné quand il se retrouve dans un état d'incapacité de traitement de tout message à cause d'insuffisance de ressource (V. Hilt, 2010). Dans ce cas, un mécanisme - défini par le RFC 6231 - contre la surcharge se déclenche, le serveur commence à envoyer des messages '503 Service Unavailable' à l'émetteur pour qu'il soit avisé de la situation. Toutefois, ce mécanisme reste relativement faible dans le contrôle de la performance, c'est ce qui a motivé plusieurs travaux de recherche à trouver des solutions plus adéquates.

1. Problématique

Parmi les premières conséquences d'une congestion d'un serveur SIP se trouve l'augmentation des délais de traitement des messages. En fait, les messages passent plusieurs secondes dans les files d'attente. Les UAC et UAS empirent la situation en retransmettant les messages plusieurs fois par UDP, le protocole de transport utilisé.

La retransmission des messages peut avoir des effets négatifs sur la qualité de service. La file d'attente se remplit par des messages dupliqués et inutiles qui ne servent qu'à générer des délais et causer des rejets inacceptables, Par conséquent nous avons une situation critique menant à une QoS très faible.

Dans ce mémoire, l'accent est mis sur l'application du protocole SIP dans la téléphonie sur IP. Les serveurs étudiés sont purement téléphoniques. On considère qu'il existe des délais d'attente plus tolérables que d'autres dans la même session SIP. À titre d'exemple : lorsque Alice essaie d'atteindre Bob par téléphone, sa tolérance d'attente pour une sonnerie (délai entre le message 'Invite' et '180 Ring') et plus grande que l'attente entre la fin de la sonnerie et la voix de Bob (délai entre le message '200 OK' et 'Ack'). Cela veut dire que lorsque Bob décroche le téléphone, l'échange des paquets voix devrait se faire le plus rapidement possible, ce qui n'est pas vraiment évident dans le cas d'un serveur congestionné.

Jusqu'à date, deux problèmes sont soulevés en cas de congestion : d'une part la retransmission des messages causés par les délais de réponse, d'autre part les délais critiques

de message qui dépassent le temps tolérable. Le troisième problème concerne les appels rejetés. On considère que pendant la période de rafale, il peut y avoir trois types de rejets : des rejets moyennement acceptables des messages 'Invite' qui sont: *peu acceptable*, les rejets des messages '180 Ring', '200 OK' et 'Ack' qui sont *inacceptable* et finalement les rejets des messages 'Bye' et '200' est qui sont *tolérable*. Ceci se résume à accepter en priorité les messages critiques des appels en cours plutôt que les messages d'établissement de nouveaux appels.

Naturellement, le rejet d'un nouvel appel est moins critique par rapport au rejet du message '200 OK'. Concrètement, si Alice essaie de contacter Bob durant la rafale. Deux possibilités s'imposent : soit son message 'Invite' est rejeté, dans ce cas elle devrait entendre la tonalité 'Occupé' sans que Bob soit au courant, soit que son message 'Invite' est traité, mais que le '200 OK' est rejeté, dans ce cas, Alice continuera à entendre la tonalité de la sonnerie même si Bob a décroché. Ce qui n'est pas vraiment acceptable. Entre temps, le message '180 Ring' est reçu, Bob décroche, mais il entend la tonalité 'Occupé' parce que le message '200 OK' a été rejeté. La situation est donc inacceptable.

Tous ces éléments problématiques sont susceptibles d'être présents dans le cas d'un serveur SIP congestionné. La façon dont les messages SIP se traitent durant la rafale est donc problématique.

2. Objectif du travail

Ce projet vise à améliorer la capacité d'un serveur SIP à travers une architecture expérimentale de test capable de remédier à tous les problèmes décrits dans la problématique. L'objectif est donc de réduire les délais critiques, éviter le rejet des messages importants avec une diminution relative de nombre d'appels rejetés.

Un système de test sera élaboré avec la combinaison de plusieurs solutions menant à résoudre la problématique. Un gestionnaire de flux est conçu afin d'évaluer l'effet de la gestion des messages sur un système SIP. La gestion des messages inclut une approche

d'ordonnancement qui sera étudiée minutieusement afin de conclure l'effet des disciplines des files d'attente sur les délais et les rejets.

3. Méthodologie et technologie:

Plusieurs travaux ont traité l'évaluation de la performance des systèmes SIP suivant des méthodologies différentes. Généralement ce projet s'inspire du processus d'ingénierie de performance SIP proposé dans (Hrischuk, 2006), ainsi que la méthode de test de performance présenté dans (Erich Nahum, 2007), toutefois il sont adaptés selon les besoins. La méthodologie suivante est utilisée :

- programmer un serveur SIP suivant la spécification RFC 3261 (Java, SQL);
- élaborer les modèles d'interaction entre différentes entités du système de test;
- élaborer les scénarios des générateurs de trafic (XML, SIPp);
- proposer et implémenter une plateforme capable de gérer le flux SIP afin d'atteindre les objectifs décrits ci-dessus (Java);
- congestionner le système proposé et comparer les résultats.

L'architecture proposée comprend un module clé responsable de la gestion de tous les messages, il sera placé devant le serveur SIP afin d'éviter sa congestion.

4. Motivation

De nos jours, SIP est largement utilisé par l'industrie de télécommunication, son application englobe une panoplie d'applications qui exige la QoS. De plus, la congestion dans le réseau SIP est un sujet qui n'est pas encore complètement défini, certains problèmes qui lui sont reliés sont incontournables et les conséquences qui en résultent sont dramatique, ce qui suscite un intérêt particulier sur la congestion de ce protocole.

5. Contribution

Le projet a pour but d'apporter une certaine amélioration au niveau de la QoS dans le cas d'un serveur SIP qui reçoit une rafale de trafic. Pour ce faire, une plateforme de gestion de flux avec six files d'attente a été élaborée.

Le système combine des méthodes de protection contre les effets de la congestion du serveur SIP. On propose une architecture expérimentale de test SIP capable d'ignorer les messages retransmis ainsi qu'une vérification continuelle de la taille des files attente pour des fins de contrôle d'admission. Ceci est combiné avec trois disciplines appliquées sur les files d'attente mises en place.

Un gestionnaire de flux SIP (GFSIP) sera mis en place afin d'effectuer un traitement différencié de message. Le GFSIP a permis d'obtenir un nouveau scénario d'échange de message entre les différentes entités SIP.

En d'autres termes, la contribution majeure consiste dans la combinaison des éléments mentionnés ci-haut avec une comparaison entre trois disciplines appliquées sur les files d'attente en question. Le résultat devrait conclure l'effet de ces disciplines sur le délai 'Invite-180', le délai '200-Ack' et les appels rejetés.

6. Structure du mémoire

Ce mémoire est structuré comme suit :

Le chapitre 1 présente un bref aperçu de SIP. Les composants, ainsi que tous les éléments de base d'un réseau SIP opérationnel y sont décrits. Des scénarios basiques utilisés dans quelques services sont illustrés. Par la suite, une synthèse sur la congestion du protocole est élaborée en se basant sur les travaux de recherches antérieures.

Le chapitre 2 décrit l'état de l'art du projet. Il commence par une description détaillée des éléments de la problématique. Il montre la relation entre les messages SIP et la tolérance au délai et au rejet. Par la suite il donne une idée de l'approche de gestion de messages proposée. Finalement, il met en évidence le rôle des disciplines appliquées sur les files d'attente dans la gestion des messages.

Le chapitre 3 présente la topologie et l'architecture de l'environnement expérimental. Le chapitre fait appel aux modèles de Markov pour assimiler le comportement des entités SIP ayant l'intention d'établir des sessions médias. Ces modèles ont facilité l'illustration des tolérances selon les différents états et transactions. L'architecture globale du test a été présentée afin de connaître les éléments SIP qui seront utilisés dans le test. Le chapitre se termine par une présentation détaillée de l'architecture du GFSIP.

Le chapitre 4 décrit les scénarios d'échange de messages entre les différentes entités qui composent le réseau de test SIP. Par la suite, l'implémentation de chaque entité est présentée d'une façon détaillée en expliquant les parties essentielles du code et les algorithmes utilisés. La méthode de déploiement et les limitations de l'implémentation sont mises en évidence à la fin du chapitre.

Le chapitre 5 donne une description détaillée des tests qui ont été effectués, suivi d'une analyse approfondie des résultats obtenus. L'analyse a ciblé la variation des délais et des rejets dans trois systèmes FIFO-GFSIP, FQ-GFSIP et PQ-GFSIP. Une comparaison entre les différents systèmes a été effectuée dans le but de conclure l'effet des disciplines sur la performance du réseau, lorsque ceci reçoit des rafales de trafic.

CHAPITRE 1

SESSION INITIATION PROTOCOL

De nos jours, la croissance des opérateurs de télécommunication est basée, essentiellement, sur la diversité des services offerts. Au départ, les opérateurs pensaient d'utiliser différents réseaux pour chacun des trafics : donnée, voix et télévision, parce que les caractéristiques des trafics diffèrent l'un de l'autre. Le flux donné tolère le délai, mais pas la perte, tandis que la voix peut tolérer la perte, mais pas le délai.

Par exemple, les réseaux ATM et Frame Relay sont des réseaux de donnée à commutation par circuit dont le taux d'erreur est très faible, toutefois, le délai est le prix à payer. De plus, la réservation des ressources posait beaucoup de problèmes de bande passante. Ces types de réseau n'étaient donc pas convenables pour supporter le trafic voix. L'internet a pu résoudre ce type de problème. Sa technologie a permis de rassembler plusieurs types de trafic dans un seul réseau. Le 'Best effort' a résolu carrément les problèmes de la bande passante (Kadoch, 2008b).

La voix est parmi les grands avantages dont l'usager a bénéficié. Les gens, à l'époque, dépensaient des prix exorbitants pour des communications longues distances. L'internet a rendu ces communications pratiquement gratuites.

La voix sur IP (VOIP: Voice Over IP) est la technologie utilisée pour faire circuler le trafic voix dans le réseau internet. Son développement nécessite un protocole permettant l'établissement et la terminaison des appels. Ce type de protocole est intitulé: protocole de signalisation.

SIP est un protocole de signalisation utilisé dans la voix sur IP. Il permet l'établissement ainsi que la terminaison des appels voix. Sa simplicité et flexibilité lui ont permis d'être adopté

dans les réseaux de nouvelles générations. À part la voix, SIP est, aussi, utilisé dans d'autres services tels que la messagerie instantanée et la présence.

SIP a été approuvé par l'IETF (Internet Engineering Task Force) en 1999 sous le RFC 2543. Son développement a eu un grand intérêt, ce qui a permis la naissance de deux groupes principaux SIP WG et SIPPING, qui ont fait un travail considérable dans la mise en marche de ce protocole (Henry Sinnreich, 2001).

1.1 Les éléments d'un réseau SIP

SIP est un protocole basé sur un code texte. Son principe ressemble à celui du protocole HTTP. Un client SIP génère une requête, le serveur répond par un message SIP. Les deux entités peuvent jouer le rôle du client et serveur dépendamment du message émis. Cette partie du mémoire présente un berf aperçu sur le fonctionnement d'un réseau SIP. Les notions discutées ont été prises de (Kadoch, 2008a).

1.1.1 Les UAs (User Agent)

Les UAs sont les terminaux d'extrémités ayant l'intention d'établir une session média. Ils peuvent être sous une forme matérielle (appareil téléphonique), comme ils peuvent être sous une forme logicielle (application). L'UAC (User Agent Client) est l'agent qui initie l'appel, tandis que l'UAS est l'agent qui répond à la demande. Les deux agents s'échangent les messages menant à l'établissement et la terminaison de la session. L'UAC peut, à un instant donné, devenir un UAS, tout dépendant du message à transmettre. Les messages, dans la plupart des cas, passent par des serveurs SIP.

1.1.2 Les serveurs SIP

Les serveurs SIP sont des machines intermédiaires dans un réseau SIP. Ils jouent le rôle d'un assistant de connexion. Leurs fonctionnalités les divisent en trois types principaux :

- serveur mandataire (Proxy): son rôle est de recevoir les messages SIP et les transférer au prochain nœud, que ce soit un UA ou serveur. Il peut être implémenté selon deux modes : 'Stateful' et 'Stateless';

- serveur d'enregistrement (Registrar Server) : son rôle est de recevoir les requêtes d'enregistrement de la part des UA et d'enregistrer/mettre à jour les informations de l'usager dans une base de données;

- serveur de redirection (Redirect Server) : son rôle est de recevoir les messages SIP et de retourner une réponse de redirection, en d'autres termes : ce serveur n'est pas chargé de transmettre le message reçu, mais il répond l'émetteur par un message contenant le contact où le message devrait être envoyé.

1.1.3 Les serveurs de localisation (Location Server)

Les serveurs de localisation sont les bases de données qui contiennent les informations sur les usagers telles que les URI, adresses IP, ou encore les informations sur d'autres éléments SIP tels que la durée d'enregistrement, adresses d'autres serveurs. Les serveurs SIP consultent les serveurs de localisation afin de récupérer les informations nécessaires permettant l'acheminement des messages.

1.2 Les messages SIP

Les messages échangés dans un réseau SIP sont sous forme de requêtes (méthodes) et réponses. Certains messages dépendent du service offert. Ceci est un exemple de requêtes utilisé dans un service voix (Henry Sinnreich, 2001). :

- REGISTER : est utilisé pour enregistrer une adresse IP auprès d'un serveur SIP;
- INVITE: indique une demande d'initiation d'une session média;
- ACK: confirme la réception d'une réponse finale du message 'Invite' qui a été envoyé;
- CANCEL: sert à annuler une requête;

- BYE: indique la terminaison de la session établie;
- OPTIONS: demande des informations sur les '*capabilities*'.

Les réponses SIP ont une forme numérique (Henry Sinnreich, 2001) :

- 1xx: Provisional: la requête est bien reçu et en cour de traitement;
- 2xx: Success: l'action est bien reçue, bien comprise et acceptée;
- 3xx: Redirection: d'autres actions sont requises pour traiter cette demande;
- 4xx: Client Error: la requête contient des erreurs de syntaxe et ne peut pas être traitée par le serveur;
- 5xx: Server Error: le serveur est dans l'incapacité de traiter cette requête même si elle est valide;
- 6xx: Global Failure: impossible de traiter la requête par n'importe quel serveur.

Les messages SIP sont acheminés d'une façon indépendante dans le réseau à travers UDP. Chacun se compose de trois éléments essentiels : la première ligne 'First line', l'en-tête du message et le corps du message (iptel, 2011). La figure 1 présente le contenu du message 'Invite'.

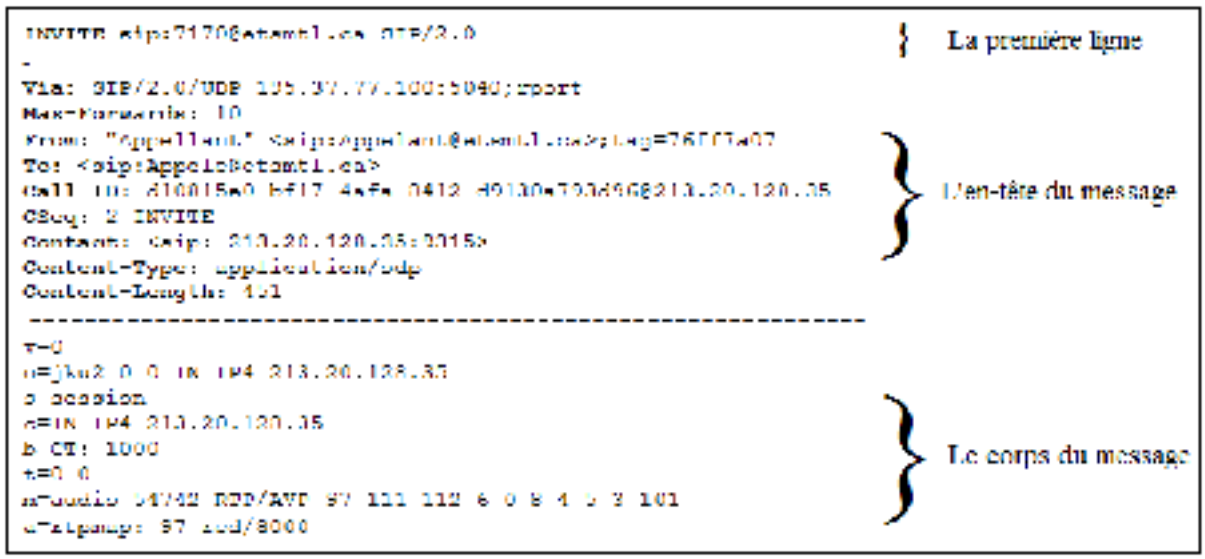

Figure 1.1 Le message 'Invite'

La première ligne désigne le type de message (requête/réponse). Dans ce cas, l'appelant initie un appel à travers un massage 'Invite'. L'en-tête du message contient des champs qui servent principalement à l'acheminement du message. Les champs '*From*' et '*To*' contiennent respectivement l'adresse de l'appelant et l'appelé. Le Call-ID est utilisé pour identifier les messages qui appartiennent au même appel. Le champ '*Contact*' contient l'adresse IP et le numéro de port auxquels il est joignable. Le corps du message contient les paramètres de la session média.

1.3 L'adressage

Les serveurs SIP sont responsables de la localisation des utilisateurs. Les UAs ne sont pas censés connaître l'adresse IP du destinataire. Dans certains scénarios, les UAs interrogent le serveur DNS (Domain Name Server) afin de récupérer une adresse IP. Toutefois, cette adresse est, souvent, celle du serveur SIP le plus prêt.

SIP utilise l'URI (Universal Resource Identifier défini dans le RFC 2396) pour identifier l'appelant et l'appelé à travers les champs 'From' et 'To' inclus dans le message reçu. Ce mode d'adressage et similaire aux adresses de courriels. Afin d'acheminer un message à un URI donnée, l'UA devrait être enregistré auprès d'un serveur SIP.

L'URI se dote de certaines flexibilités permettant de faciliter la connexion avec le réseau PSTN. L'adresse `sip:514-885-2283@etsmtl.ca;user=phone` indique que l'appel est destiné au réseau PSTN et qu'il doit passer par la passerelle de téléphonie IP ayant le nom de domaine `etsmtl.ca`. Le 'Tag' `phone` montre qu'un numéro de téléphone est dans la partie nom utilisateur de l'URI.

1.4 Le dialogue

SIP est un protocole utilisé dans une panoplie de services dont les messages et les scénarios dépendent intimement. Cette partie présente des scénarios basiques de trois services (voix,

messagerie instantanée et présence) qui utilisent SIP dans l'établissement et la terminaison des sessions.

La figure 1.2 présente un scénario basique d'enregistrement et d'initiation d'un appel. Les opérations d'enregistrement s'effectuent périodiquement par le téléphone sans l'intervention de l'être humain. Un champ 'Expires' dans l'en-tête du message 'Register' définit le temps d'expiration de l'enregistrement. Les deux UA devraient être préalablement enregistrés avant de pouvoir effectuer les appels. Le serveur SIP sera, donc, en mesure de localiser la machine ou l'URI en question est connecté.

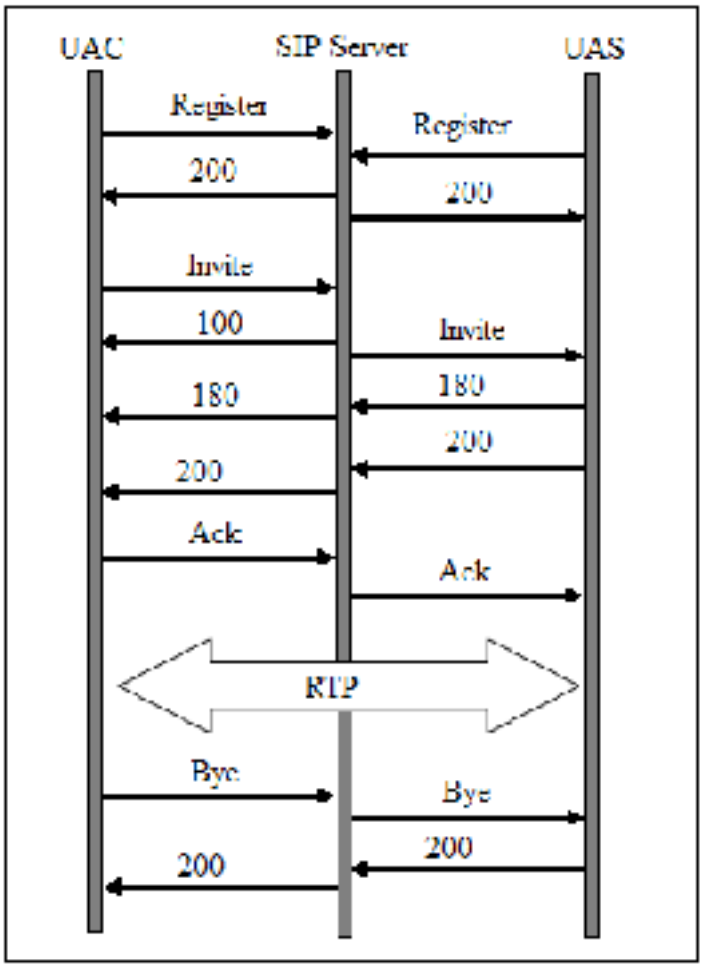

Figure 1.2 La signalisation SIP dans un appel VOIP

Afin d'initier un appel, l'UAC envoie un message 'Invite' contenant l'adresse du destinataire. Ce message contient souvent les informations de la session média. Le serveur SIP reçoit le message, consulte l'adresse de destination et transmet le message au UAS. Ce dernier répond par un message '180 Ring' si le téléphone n'est pas occupé. Lorsque l'appelé décroche, un message '200 OK' est envoyé au serveur SIP qui le transmet à l'UAC. Ceci confirme par un message 'Ack'. La session RTP est établie une fois le UAS reçoit ce message. À la fin de la conversation, les deux messages 'Bye' et '200 OK' sont échangés afin de mettre fin à la session courante.

L'échange de message peut différer dans chaque implémentation. Il est possible que les messages 'Ack' , 'Bye' et '200 Ok (Bye)' s'échangent directement entre les UAs. D'autres implémentations préfèrent passer tous les messages par un serveur SIP pour des raisons de sécurité et de contrôle. Par exemple: le passage des messages 'Bye' par un serveur SIP pourrait déterminer la durée de chaque appel.

Le protocole SIP est aussi utilisé dans la messagerie instantanée (MI). La figure 1.3 présente un scénario d'échange de message dans ce service.

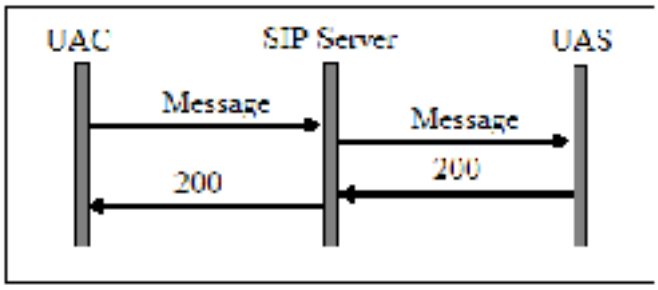

Figure 1.3 Le scénario de la messagerie instantanée

Ce service consiste en un échange de message en temps réel pour des fins de communication. Les messages échangés ont une forme texte, et sont naturellement courts. Les utilisateurs commencent une session 'Chat' durant laquelle ils s'échangent des messages texte. Le message 'Message' est utilisé pour transporter le message instantané. Il est envoyé par l'UAC, reçu par le serveur SIP qui le transmet à l'UAS. Ce dernier répond par un message

‘200 OK’ confirmant la réception du message. Le serveur SIP pourrait avoir recours à une base de données afin d’effectuer l’acheminement et la gestion des transactions.

Le service MI est souvent combiné avec le service de présence dans les applications de communication. Naturellement, les utilisateurs se connectent à une application donnée et ils se mettent au statut souhaité (Disponible, Occupé, Partis manger…). La gestion des informations de présence s’effectue selon un modèle établi dans (M. Day, 2000). Ce modèle contient trois entités logiques : *Presence Service*, ‘*Presentity’* et le ‘*Watcher’*.

Presence Service : ce service reçoit les informations de présence, l’enregistre et l’envoie aux différents UAs.

Presentity : est le UA qui fournit l’information de présence au ‘*Presence Service*’ pour qu’elle soit distribuée aux différents UAs.

Watcher : l’entité (UAS) qui reçoit l’information de présence

La figure 1.4 présente un exemple d’échange de message dans un service de présence.

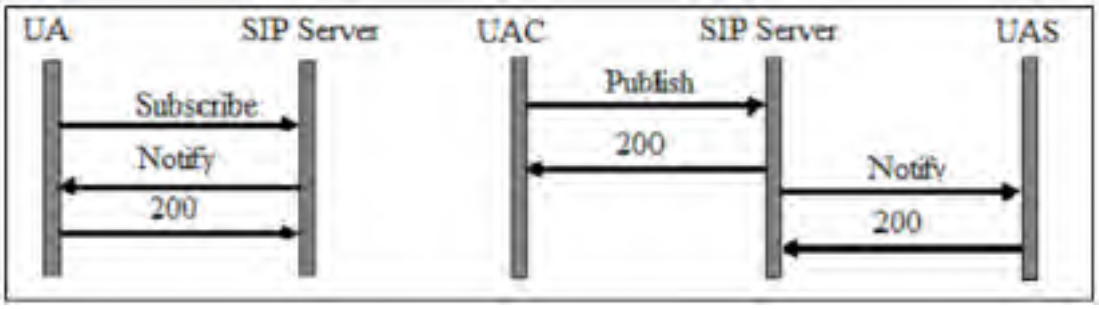

Figure 1.4 Les scénarios de Présence

Le serveur SIP est responsable sur l’échange des messages ainsi que la mise à jour des statuts des utilisateurs dans la base de données. Il joue le rôle du ‘*Presence Service’*. L’UA (*Subscriber*) demande l’information de présence auprès du serveur SIP qui l’envoie à travers

le message ‘Notify’. Les changements d’information de présence sont des actions déclenchées par les utilisateurs. Le protocole SIP se sert du message ‘Publish’ pour notifier le serveur SIP (*Presence Service)* du changement. Ce dernier transmet l’information aux UAs concernés à travers le message ‘Notify’.

Les informations fournies jusqu'à date donnent une idée générale sur le fonctionnement du protocole SIP. La partie qui suit introduira un des problèmes le plus courent : la congestion.

1.5 La congestion dans les Réseaux SIP

La congestion, en général, peut ‘frapper’ plusieurs composants d’un réseau, notamment les routeurs dans le cas d’un réseau IP. En conséquence, le trafic subit un ralentissement et sera rejeté lorsque les tampons seront pleins. Lorsque le trafic SIP est présent dans un réseau congestionné, les appels seront de mauvaise qualité. Les délais seront inacceptables, et les appels seront rejetés dans la plupart du temps. Cela est frustrant pour un client qui paie sa facture mensuellement.

Les serveurs SIP, tout comme les autres serveurs, peuvent être congestionnés. Le nombre de messages élevé qu’un serveur reçoit peut dépasser ses capacités. Lorsque le protocole UDP est utilisé, la retransmission des messages perdus, lorsque les tampons sont pleins, multiplie le nombre de messages reçus par le serveur. Ce qui résulte en une dégradation de performance.

La qualité de service exige que le serveur SIP soit toujours disponible à traiter les requêtes sans générer de délai inacceptable. D’où les mécanismes de protection contre la congestion doivent impérativement être présents dans n’importe quel réseau SIP.

Afin de protéger un serveur SIP, les sources de la congestion doivent être identifiée, ceci lui permettra de réagir convenablement. Il est aussi important d’établir des modules de surveillance afin d’éviter l’interruption complète du service, ce qui est fatal pour un opérateur de télécommunication (Sisalem, 2011).

1.5.1 Les causes de la congestion dans les réseaux SIP

Plusieurs causes peuvent engendrer la surcharge d'un serveur SIP. Ils peuvent être sous forme d'attaque malicieuse telle que les Dénis de service, où l'attaquant génère un très grand nombre de requêtes, qui sont dans la plupart des cas invalides et consomment un grand pourcentage du CPU. Le serveur se trouve souvent dans l'incapacité de traiter les messages valides.

Les rafales de trafic qui arrivent d'une façon inopinée et qui ne durent pas longtemps, sont aussi parmi les causes de la congestion. Ce scénario peut être le cas d'un centre d'appel qui reçoit un nombre très grand d'appels à cause d'une panne quelque part dans un système. Le nombre d'appels reçus par les serveurs VOIP, dans ce cas, est souvent au-delà de leurs capacités.

Les erreurs logicielles et de configuration peuvent aussi engendrer un état de congestion. Une boucle infinie peut limiter la capacité du serveur, le nombre de messages dont il est capable de traiter sera réduit. En conséquence, même si le nombre de messages reçus n'est pas trop élevé, le serveur se trouvera dans l'incapacité de les traiter. D'autre part, le serveur peut être mal configuré de telle sorte qu'il multiplie le nombre de messages à envoyer. En résultat : le serveur voisin sera immergé de message. Le risque peut être propagé à d'autres serveurs, ce qui produit une dégradation de performance dans tout le réseau. Les informations sur les causes de congestion ont été prises de la source (Sisalem, 2011).

1.5.2 Le contrôle de la congestion dans les spécifications SIP

Le RFC 3261 n'a pas donné assez d'importance à la congestion. Les travaux de recherche qui ont été effectués à cet égard décrivent le mécanisme comme étant faible. Selon le RFC en question, lorsqu'un serveur est congestionné, il peut rejeter les messages en utilisant le message '503 Service Unavailable'. Le serveur répond par ce message afin d'aviser le transmetteur de son état. Le message '503' peut comprendre un champ 'Retry-After'

indiquant la période de temps que le transmetteur doit attendre avant d'envoyer le message. Ce mécanisme peut engendrer une propagation de la congestion, puisque les messages seront redirigés vers d'autres serveurs.

Lorsqu'un réseau SIP contient un seul serveur SIP, les UA n'ont pas d'autre choix que diriger leurs messages vers ce serveur. Le rejet d'appel devient critique dans ce cas, les usagers ne sont, souvent, pas compréhensible, et continuent à appeler. En résultat : la situation s'empire de plus en plus.

1.5.3 Les travaux de recherche antérieurs

Étant donné que le mécanisme de contrôle de la congestion implémenté dans le protocole SIP est faible, beaucoup de travaux de recherche ont été effectués afin de trouver d'autres solutions. La source (Sisalem, 2011) présente une synthèse sur le contrôle de congestion en se basant sur des travaux de recherche antérieurs. L'article a relevé quatre points importants dans la conception d'un mécanisme de contrôle de congestion afin d'aboutir à un système performant.

- la surveillance : comprend les informations qui doivent être surveillées afin d'indiquer l'état du serveur;
- l'algorithme : détermine la meilleure approche utilisée pour réduire la surcharge lorsque la congestion est détectée, en tenant en considération les problèmes reliés;
- la Réaction : la méthode utilisée pour réduire trafic lorsque l'algorithme a décidé de le faire;
- le modèle : détermine si le mécanisme sera exécuté dans un seul serveur ou bien, d'autres serveurs doivent être impliqués.

Les auteurs (Hilt et Widjaja, 2008) ont effectué une étude de performance d'un système SIP lorsque celui-ci est congestionné. Deux serveurs ont été mis en place avec des capacités limitées afin de simuler la congestion. Le mécanisme d'envoi du message '503 Service Unavailable' avec les champs 'Retry-After' a été appliqué. En résultat les deux serveurs étaient sévèrement congestionnés lorsque le débit a dépassé leurs capacités. Le débit de sortie a chuté carrément notamment après l'expiration du temporisateur 'Retry-After'. Les deux serveurs se sont trouvés dans l'incapacité de traiter les messages reçus.

Deux algorithmes *Bang-Bang* et *Occupancy* de contrôle de congestion local ont été comparé (Hilt et Widjaja, 2008). Les résultats ont démontré que les algorithmes de contrôle local sont capables d'atteindre le champ dans lequel le serveur opère sans perte de messages. Le serveur SIP qui implémente *Bang-Bang* a deux états : *'overload'* et ' *underload*'. Ces états sont déterminés par le nombre de message qui existe dans la file d'attente. Ainsi, lorsque ce nombre dépasse une certaine valeur maximale, le serveur se déclare '*overloaded*' et commence le rejet des appels. Sinon il reste dans son état '*underloaded*' et opère d'une façon normale. Un autre algorithme semblable a été introduit dans (Montagna et Pignolo, 2008) : CCT (Control Through Tresholds). Dans cet algorithme la taille de file d'attente a deux valeurs Min et Max. L'algorithme tiens en considération quatre scénarios différents : 1) le nombre de message est inferieur à Min, 2) le nombre de message est supérieur à Max 3) le nombre de message est entre les deux valeurs mais en augmentation 4) le même que (3 mais le nombre est en cours de diminution. Dépendamment du message et du scénario, l'algorithme détermine la décision de rejet.

L'algorithme *Occupancy* accepte les messages 'Invites' en se basant sur une probabilité *F* qui dépend de L'utilisation du CPU. Le but est d'ajuster *F* afin de maintenir l'utilisation du CPU au-dessous d'une valeur précise. La vérification du CPU se fait d'une façon périodique afin de comparer son utilisation à la valeur maximale. Ainsi, La manipulation de *F* a permis un certain contrôle sur charge du CPU.

Les deux algorithmes ne sont pas capables de prévenir la congestion. Pour résoudre le problème, un contrôle de congestion distribué a été implémenté. Ceci est basé sur des '*Feedback*' échangés entre les serveurs afin de garantir un certain lissage de trafic.

Les algorithmes de '*FeedBack'* sont généralement utilisés entre serveurs pour que le '*Upstream Server'* envoi une quantité de trafic que le '*Downstream Server'* est capable de traiter. Trois algorithmes de ce type ont été proposés dans (Shen, Schulzrinne et Nahum, 2008) : Win-*disc*, *Win-auto* et *Win-cont*. Ces algorithmes se basent sur l'ajustement d'une fenêtre qui détermine le volume de trafic à envoyer. Dans chacun des cas, la taille de la fenêtre est déterminée d'une façon différente en se basant sur l'information échangée entre les serveurs. Ceci, bien évidemment, permettra au '*Upstream Server'* de lisser le trafic envoyé au '*Downstream Server*'.

Le *Win-disc* (*window-discrete*) est un algorithme qui calcule le nombre de session SIP que le serveur peut accepter à la fin de chaque intervalle de temps discret. Ce calcule tiens en considération le temps de traitement de message en cours et ceux qui arrivent. Et donc, à la fin de chaque période de contrôle, le serveur devrait recevoir le nombre de message qui a été calculé dans la période précédente. Cet algorithme est basé sur des intervalles de temps bien précis, Ce qui change en permanence la valeur de la fenêtre envoyé comme '*FeedBack'*.

Contrairement au *Win-disc*, Le calcule de la taille de la fenêtre dans *Win-auto (window-continuous*) est basé sur des événements. Et donc, le calcule ne se fait pas chaque intervalle de temps mais dépendamment de l'événement déclenché. Dans n'importe qu'elle moment, la valeur de la fenêtre est calculé puis envoyé comme '*FeedBack'* au *'upstream Server'*. Dans cet algorithme la valeur de la fenêtre est la différence entre le nombre maximale de session permis et le nombre de session en cours.

Le *Win-auto (window-autonomous*) a pour but d'ajuster la valeur de la fenêtre d'une façon autonome, de telle sorte que le taux d'augmentation de la taille de la fenêtre soit **toujours**

plus faible que celui de sa diminution. Ainsi le débit d'arrivé restera relativement faible par rapport au débit de service.

En général les algorithmes de contrôle de congestion peuvent être implicites ou explicites (V. Hilt, 2010). Un contrôle explicite est utilisé par un serveur SIP pour indiquer explicitement que sa capacité a atteint ses limites. Les serveurs qui reçoivent cette information doivent baisser le débit de transmission. Ceci est une liste de politique de contrôle de congestion à base d'un '*Feedback*' explicite (V. Hilt, 2010) :

- Rate-based Overload Control : Dans ce type d'algorithme, le serveur surchargé calcule la valeur de débit de message qui peut supporter, cette information est passé au '*Upstream Server*' afin d'être notifié, ce qui permet de limiter le nombre de message.
- Loss-based Overload Control: ce type de contrôle surveille régulièrement le tau de perte de message. Lorsque ceci dépasse une certaine limite, le serveur en question demande au '*Upstream serveur'* de réduire le nombre de message.
- Windows-based Overload Control: les algorithmes à base de fenêtre se basent sur la taille de la fenêtre, celle-ci dépend de la l'information du '*FeedBack*'. Naturellement, cette information est déterminée dépendamment de l'algorithme utilisé (i.e Win-*disc*, *Win-auto* et *Win-cont*).
- Overload signal based Control: ce contrôle se base sur le message 503 (Service Unavailable). Un serveur qui reçoit un tel message continue à réduire le débit de sortie jusqu'à ce qu'il n'y a plus de rejet.
- On-/off Overload Control: le serveur qui implémente ce type de contrôle a deux état : ON : le serveur est capable de recevoir les messages, OFF : le serveur est dans l'incapacité de recevoir les messages.

Dans un contrôle implicite, les serveurs utilisent les délais et la perte des paquets pour déduire la congestion. Le serveur congestionné n'a pas à aviser les serveurs voisins, mais ceux-ci sont dotés de mécanisme permettant de déduire son état à travers son comportement.

Que ce soit un algorithme implicite ou explicite, l'effet sur l'utilisateur final (UA) sera le même. Notamment lorsque la congestion est propagée dans le réseau. En fin de compte, les serveurs de bordure sont obligés de rejeter les appels reçus directement des UAs. Une sorte de priorité d'appel peut être combinée avec les algorithmes de contrôle de congestion afin de prioriser les appels importants. Toutefois, il existe certains cas où la priorité des appels ne peut être établie. Par exemple, le cas d'une émission télévisée qui offre des cadeaux aux premiers téléspectateurs qui appellent. Dans ce cas, le serveur SIP congestionné devrait traiter tous les appels d'une façon équitable.

1.6 Conclusion

Il devient clair que le protocole SIP est un protocole de signalisation qui se base sur des messages textes, permettant de connecter deux terminaux afin d'établir des sessions médias. Les composants d'un réseau SIP ont été mis en évidence, avec des exemples de scénario d'échange de message dans le cas de trois différents services.

L'un des éléments critiques dans le réseau en question est le serveur SIP. Il a été montré que sa performance peut subir une dégradation, à cause du nombre de messages qu'il reçoit. Ce nombre dépasse souvent sa capacité, ce qui le met dans un état de congestion.

Dans ce mémoire, les solutions proposées ont été introduites d'une façon superficielle. Pour plus d'information sur l'algorithme décrit précédemment, il est recommandé d'avoir recours aux références reliées. La plupart des efforts ont été mis sur les algorithmes de contrôle de congestion entre serveurs. Tous ces algorithmes se basent sur l'information échangée (*FeedBac*k) afin de baisser le volume du trafic destiné au serveur congestionné.

La collaboration des serveurs est, donc, un élément clé dans la réussite de tous ces algorithmes. Maintenant la question qui se pose est : comment peut-on améliorer la qualité de service d'un réseau SIP qui contient un seul serveur SIP, lorsque le débit du trafic reçu est très élevé ? Bien évidemment, tous les algorithmes vus précédemment ne peuvent être appliqués. Le chapitre suivant présente l'état de l'art en réponse à cette question.

CHAPITRE 2

L'ÉTAT DE L'ART

La qualité de service dans les réseaux SIP exige une performance continue des serveurs responsables du traitement des requêtes. Naturellement ces serveurs jouent le rôle d'un intermédiaire entre les entités communicantes. Lorsqu'ils tombent en panne, tout le réseau risque de tomber en panne.

Il existe une différence entre un serveur congestionné et un serveur qui reçoit une rafale de trafic (High-load situation) (Batteram, Meeuwissen et Bemmel, 2006). Dans le cas d'une rafale, le nombre de messages reçus par le serveur SIP est au-delà de sa capacité de traitement pour une période de temps limitée. Toutefois, il tient toujours le contrôle sur les messages reçus. Par contre un serveur congestionné est un serveur qui est temporairement et partiellement en panne parce que le trafic reçu a dépassé sa capacité. Les messages, dans ce cas, se perdent hors-contrôle du serveur. Naturellement, le débit de sortie, dans ce cas, chute dramatiquement (Hilt et Widjaja, 2008) (Shen, Schulzrinne et Nahum, 2008). Dans ce mémoire, la phrase 'serveur congestionné ou surchargé' peut être utilisée pour décrire l'état du serveur lorsque les files d'attente sont pleines, même s'il a toujours le contrôle sur les appels rejeté. À cet égard, il est essentiel de tenir en considération que ce mémoire limite le champ d'études sur la période de rafale. La congestion du serveur SIP - SIP Overload selon (Batteram, Meeuwissen et Bemmel, 2006) - n'est pas prise en considération.

SIP est un protocole qui se base sur des messages pour établir les sessions. Le volume de ces messages est l'élément problématique responsable sur la dégradation de la performance. Naturellement, lorsqu'un serveur SIP reçoit une rafale de trafic, il traite les messages selon sa capacité. En conséquence, les délais et les rejets deviennent de plus en plus importants. Des méthodes de gestion de message sont donc nécessaires pour profiter de la capacité des serveurs. En résultat : les délais et les rejets devraient être minimisés.

2.1 La tolérance aux délais des messages SIP

L'échange de message dans le protocole SIP est organisé sous forme de session. La signalisation dans chaque session est sous forme de plusieurs étapes. La tolérance au délai est différente dans chaque étape. Lorsqu'un usager initie un appel, il est tolérable à un délai de quelques secondes avant d'entendre la tonalité de la sonnerie. Toutefois, si ce temps dépasse une certaine limite, l'appel devrait être accepté. En d'autres termes, l'appelant ne devrait pas attendre longtemps lorsque le système rejette son appel, l'appelant devrait recevoir la tonalité 'occupé' tout de suite. Le délai entre la fin de la tonalité et le début de la tonalité de la sonnerie du coté appelant est considéré : *Peu acceptable.* Dans ce mémoire, ce délai est appelé : 'Invite-180'.

Lorsque l'appelé décroche, la conversation doit commencer tout de suite, et donc, les messages échangés durant cette étape devraient être traités rapidement. Naturellement la session RTP commence juste après la réception des messages '200 OK' et 'Ack' par les UAs. Le délai de ces deux messages est appelé : '200-Ack', et il est considéré : *Inacceptable*.

À la fin de la conversation, les deux messages 'Bye' et '200 OK' sont échangés afin de mettre fin à la session SIP courante. L'usager raccroche le téléphone sans se préoccuper du délai de la signalisation qui s'effectue juste après. Ce délai est, donc, considéré : *Tolérable.*

La tolérance au délai dépend de l'étape de signalisation dans une session SIP. Un traitement particulier devrait s'effectuer dépendamment du message. Ainsi, la gestion des messages peut être implémentée afin de respecter cette tolérance, notamment lorsqu'il est possible de profiter de quelque tolérance en faveur des autres. Plus de détail sur ces délais seront analysés dans le chapitre 3, ou des modèles de Markov seront introduits.

2.2 Les rejets et la priorité des messages SIP

L'approche proposée donne différents poids aux messages SIP parce que chacun a une tolérance différente. Le serveur SIP reçoit les messages selon un ordre préétabli. Cet ordre

sera effectué par un mécanisme de gestion de message. Le rejet s'effectue dépendamment du message, la tolérance au rejet est semblable à celle du délai. Trois types de rejet seront analysés pour des fins de comparaison : les rejets *'acceptables'*, '*peu acceptables'* et '*tolérables'*.

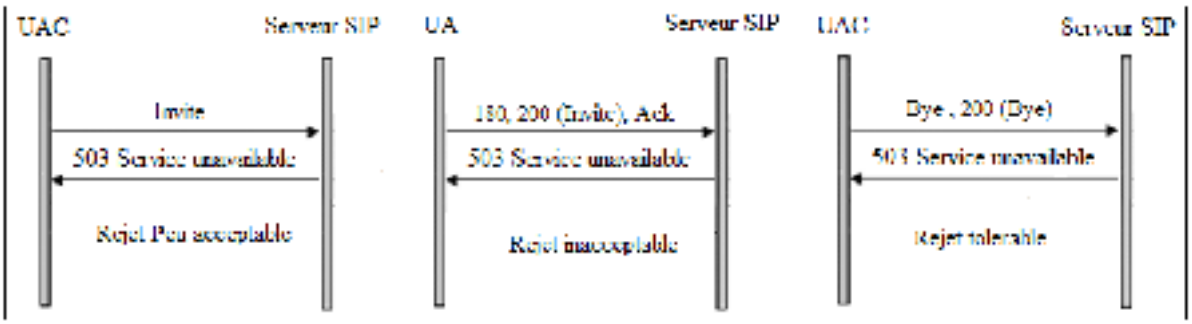

Figure 2.1 Les types de rejet

La figure ci-dessus montre les scénarios des trois types de rejet qui existent. Dans chacun des cas L'UA émet un message SIP au serveur SIP. Selon l'étape de signalisation, le serveur SIP répond en tout temps par un message '*503 Service Unavailable*'.

Il devient clair que les types de rejets établis nécessitent une certaine priorité au niveau des messages SIP. Le critère de priorité des messages se base sur l'étape de signalisation. En d'autres termes, plus on s'engage dans la signalisation d'établissement d'appel plus la priorité du message devient élevée. Ainsi, le traitement des messages 'Ack' est plus prioritaire que '200 OK (Invite)' ce dernier est plus prioritaire que '180 Ring'.Le message 'Invite' est moins prioritaire que le '180 Ring'. Finalement, les messages 'Bye' et '200 Bye' ont la priorité la plus basse.

Il faut tenir en considération que dans la réalité, la rafale de trafic voix qui arrive tout à la fois ne dure pas longtemps. Et donc, effectuer des rejets pour des messages à basse priorité sera momentané. Dès que le trafic reprend sa nature normale, il n'y aura plus de rejet.

La basse priorité affectée aux messages ‘Bye’ engendre un certain retard de traitement durant la rafale. Ainsi, le mécanisme de gestion de message peut soit, effectuer un rejet soit ignorer le message. Le choix entre ces deux opérations est très important durant la congestion, car il est considéré, que le coût du rejet est non-négligeable par rapport au coût de traitement des autres messages (Shen, Schulzrinne et Nahum, 2008). Durant la rafale, les messages à basse priorité ne seront pas traités dès leur réception. Si la durée de rafale est courte, ces messages vont être traités juste après. Sinon, il sera trop tard de les traiter suivant les spécifications du (J.Rosenberg, 2002) et donc, ils subiront un traitement local et particulier : ils seront enregistrés pour garder trace sur les informations des appels. Ainsi, il sera possible de mémoriser la durée de chaque appel. En ce qui concerne les UAs, ils doivent, automatiquement, se mettre à l’état ‘Terminé’ si aucun message ‘Bye’ n’est reçu, ou bien lorsque l’appelé raccroche. La qualité de service exige, donc, une priorité minime pour les messages de terminaison d’appel, et priorise les messages d’établissement d’appel. Cette opération aura un grand impact sur le nombre d’appels admis.

La priorité des messages SIP n’est pas une approche nouvelle. Un travail considérable a été effectué par (Batteram, Meeuwissen et Bemmel, 2006). Les auteurs ont développé un module qui détermine la priorité du message et le met dans une file d’attente prioritaire. Les requêtes et les réponses ont eu différentes priorités. L’étude de la priorité a été effectuée d’une façon générale, ainsi, dans un seul réseau SIP, il existe des messages ‘Invite’ de priorité différente. Et donc, les appels ont des priorités différentes. Cela ressemble, plus en moins, au travail effectué par les auteurs de (Jing, Hongliang et Weimin, 2008).

Le principe de priorité dans ce mémoire est différent. D’une part, les appels ont tous la même priorité. D'autre part, la méthode appliquée est différente. Le mémoire se base sur un principe de discipline effectué entre différents messages d’une session SIP. Il se peut que le résultat soit le même, mais c’est une autre façon de confirmer l’importance de la différentiation entre les messages SIP. Comme il a été démontré dans les différents travaux (Jing, Hongliang et Weimin, 2008; Jing et al., 2007; Jing et al., 2009).

Jusqu'à date, il existe trois messages ayant une tolérance inacceptable : ‘180 Ring’, ‘200 OK (Invite)’ et ‘Ack’. Ces messages se considèrent critiques et ils se sont dotés d’une priorité différente. La priorité établie ne signifie pas qu’il y aura des délais inacceptables pour les messages critiques à basse priorité en cas de rafale. En effet, il se peut que le système reçoive tellement de messages ‘Ack’ qu’il générera des délais inacceptables pour d’autres messages critiques tels que ‘200 OK’. Toutefois, le système de gestion de message devrait être capable de contourner ce problème, et de traiter tous les messages critiques en respectant les délais inacceptables. Le prochain chapitre mettra en évidence la méthode utilisée.

2.3 La gestion des messages SIP et l’approche proposée

La gestion des messages est la méthode utilisée afin de respecter les tolérances en question. Un module de gestion flux SIP, intitulé GFSIP (Gestionnaire de flux SIP) est proposé afin de permettre la gestion des messages. Un module du même type a été proposé dans (Jing, Hongliang et Weimin, 2008; Jing et al., 2007; Jing et al., 2009). Les auteurs ont fait la conception de ce qu’ils ont appelé un FEFM (Front End Flow Management) capable de limiter la concurrence des messages dans le serveur. Ce module a été placé devant une application serveur SIP, et a permis de respecter les SLA (Service Level Agreement) établis, en priorisant les requêtes concernées. Le FEFM rejette les nouvelles requêtes quand le volume d’appel est élevé, ce qui permet de protéger les sessions qui étaient déjà en cours, et il détecte les messages retransmis, ce qui a permis une diminution considérable au niveau des délais.

Ce projet a été inspiré des travaux (Jing, Hongliang et Weimin, 2008),(Jing et al., 2007) et (Jing et al., 2009) . Le FEFM et le GFSIP ont le même objectif : la gestion du flux SIP. Toutefois, chacun implémente des algorithmes différents et traite des types de flux différents. Le GFSIP est conçu pour être placé devant un serveur SIP afin de lui transmettre les messages selon un ordre respectant les tolérances définies auparavant. Il est configuré pour respecter un seul SLA pour tous les appels. Contrairement au (Jing et al., 2007), ou ils ont considéré diffèrent SLA pour différents appels.

Le FEFM est basé sur deux files d'attente contenant les nouvelles et anciennes requêtes, dans chaque file, un temps d'attente maximale est établi. Lorsqu'un message 'Invite' est reçu, le FEFM le met dans la file d'attente des nouvelles requêtes, Un module intitulé '*throttle*' est mis en place afin de limiter le nombre de requête qui s'exécute d'une façon concurrente dans le serveur. Ce module décide si une requête peut être retirée de la file d'attente dépendamment du nombre de requête en exécution dans le serveur.

Le but du FEFM est de faire passer les requêtes qui vont satisfaire le SLA établi. La file d'attente des nouvelles requêtes est consulté d'une façon prioritaire, si le temps de réponse du message (calculé à base d'une prédiction) est inférieur au maximum, le message est envoyé au serveur, sinon le message passe à la file d'attente des anciennes requêtes. Cette file d'attente est vérifié lorsque le FEFM a la possibilité, le temps d'attente des messages qui existe dans cette file d'attente est comparé à une certaine valeur maximale spécifiques pour la file, s'il est inferieur, le message et envoyé eu serveur, sinon la session est rejetée, et donc seulement les messages 'Invite' qui existent dans la file d'attente des anciennes requêtes peuvent être rejetés s'ils dépassent une certaine limite de temps.

L'algorithme est mis en place afin de respecter les SLA en servant les deux files d'attente. Toutefois, les messages Non-Invite' échangés durant la signalisation n'ont pas été pris en considération. Cette limitation a été prise en considération par le GFSIP.

Une architecture de test sera mise en place afin d'évaluer l'effet de la gestion des messages sur les délais et les rejets. Le processus d'ingénierie de performance du système a été inspiré du travail (Hrischuk, 2006). Une plateforme d'exécution automatisée a été élaborée afin générer un trafic à haut débit et collecter les résultats. Cette plateforme est sous forme d'une architecture d'évaluation 'Benchmark' qui permet de tester les applications-SIP serveurs. L'élaboration de cette architecture a été inspirée du travail (Erich Nahum, 2007). Les technologies utilisées seront présentées dans les prochains chapitres.

2.4 Les files d'attente et les messages SIP

L'application des disciplines des files d'attente a eu un effet considérable dans la qualité de service. Ces disciplines permettent de traiter les paquets selon leurs caractéristiques. En d'autres termes, un trafic sensible au délai sera traité plus rapidement. Pour se faire, sa file d'attente sera consultée en priorité afin de récupérer les messages en attente. Naturellement la discipline appliquée affecte la latence des paquets parce qu'elle détermine le message qui va être transmis. Ceci dit que d'autres paquets subiront un temps d'attente additionnel.

Le GFSIP implémenté fournit un traitement adéquat des messages selon leurs tolérances. Il se base sur des files d'attente établies pour chaque type de message. L'étude vise la relation : tolérance versus discipline. En fin de compte, il faut démontrer l'effet des disciplines sur les tolérances définies auparavant.

L'étude des différentes disciplines dans le protocole SIP est intéressante, parce que l'échange de messages est organisé sous forme de session. Cette caractéristique suscite une certaine dépendance entre les différents messages qui sera expliquée en détail dans le dernier chapitre.

2.5 Conclusion

Les tolérances qui ont été définies concernent le même appel SIP. En d'autres termes, le GFSIP va recevoir des appels ayant la même priorité. Toutefois, les étapes de la signalisation ont des tolérances différentes. Un réseau SIP devrait être capable de prendre les tolérances en considération en effectuant un traitement particulier pour chape étapes de signalisation. Pour ce faire, le GFSIP a été implémenté en utilisant différentes files d'attente. Trois différentes disciplines seront appliquées afin de démontrer leurs effets sur les délais et les rejets.

Les prochains chapitres vont mettre en évidence les architectures utilisées pour implémenter la gestion de flux proposée. L'architecture du module responsable de la gestion de flux sera aussi présentée. Un nouveau scénario d'échange de message entre les éléments SIP sera introduit. Pour plus de détails, un des chapitres sera consacré á présenter l'implémentation de chacun des modules utilisés.

CHAPITRE 3

L'ARCHITECTURE EXPÉRIMENTALE DE TEST

Le système proposé se compose de plusieurs modules capables de tester la performance de la signalisation SIP. Il existe plusieurs architectures ayant le même but, toutefois, chacune cible un problème bien précis. Plusieurs facteurs peuvent engendrer la congestion d'un serveur SIP, mais la congestion en tant que telle se résume en deux topologies : Serveur - Serveur et UA – Serveur

3.1 Topologie : Serveur - Serveur vs UA – Serveur

Dans le cas d'une surcharge d'un serveur SIP, on fait appel à deux topologies communes : Serveur - Serveur et UA – Serveur (Hilt et Widjaja, 2008; Shen, Schulzrinne et Nahum, 2008).

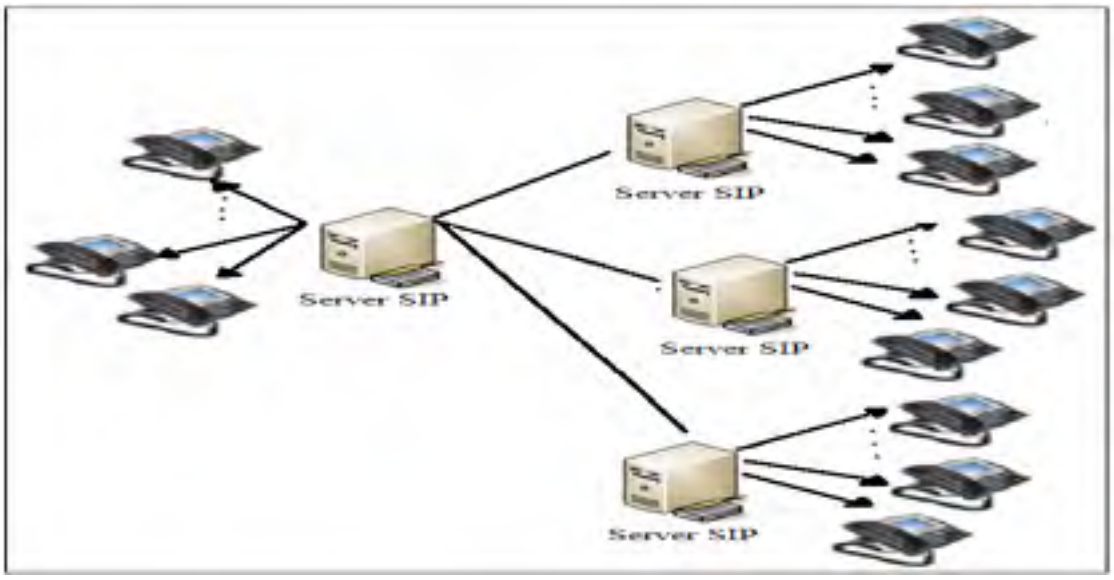

Figure 3.1 La topologie Serveur-Serveur

La figure 3.1 illustre les éléments qui constituent une topologie Serveur-Serveur. On remarque que plusieurs serveurs sont connectés afin de relier les UAC et UAS. Lorsqu'un serveur est congestionné, tous les autres éléments peuvent être affectés ce qui mène à une dégradation de performance dans tout le réseau. Les travaux de recherche mentionnés dans le chapitre 1 ont, justement, essayé de régler ce type de problème. Cette topologie et les problèmes qui en résultent sont hors de la portée de ce travail.

Figure 3.2 La topologie UA-serveur

La topologie UA-serveur (Figure 3.2) comprend une structure moins complexe par rapport à la topologie précédente. Tous les messages de signalisation doivent passer par un seul serveur SIP afin d'établir une session avec succès. Ceci reflète clairement les conséquences de la surcharge du serveur en question sur les messages échangés entre les UAC et UAS.

Contrairement à la topologie Serveur-Serveur, le seul moyen pour baisser le nombre de messages reçus est d'effectuer les rejets. Le rejet d'un nouvel appel implique le rejet du message 'Invite', et donc les cinq messages qui suivent ('180 Ring', '200 OK', 'Ack', 'Bye', '200 OK') seront évités. Ce qui permettra de diminuer les débits des messages et finalement permettra aux files d'attente de se vider.

Cette topologie existe dans des entreprises qui comptent sur un seul serveur pour connecter tous les téléphones. Et donc, plus le nombre des UAC / UAS augmente plus la probabilité de la congestion est grande. Dans la réalité, les rejets d'appel en cas de congestion ne servent

qu'à empirer la situation : les appelants deviennent frustrés et continuent à composer les numéros. Il en résulte une situation de plus en plus critique.

Ce projet discutera le problème de congestion UA-serveur. L'objectif sera basé sur cette topologie pour apporter une certaine amélioration aux éléments de la qualité de service qui ont subi une dégradation. On considère qu'il existe des moments de rafale d'appels qui arrivent tout à la fois et qui durent pendant une période de temps bien précise. La rafale produira des délais inacceptables que le système doit contourner afin de satisfaire les exigences de la QoS.

3.2 L'architecture standard de test et Modules utilisés

Cette partie montre les différents modules composant un système standard d'évaluation SIP suivant une topologie UA-serveur (Figure 3.3). Les appels sont générés par un générateur de trafic qui, d'une part, simule le comportement humain (utilisateur) et d'autre part simule le comportement du téléphone (application/appareil).

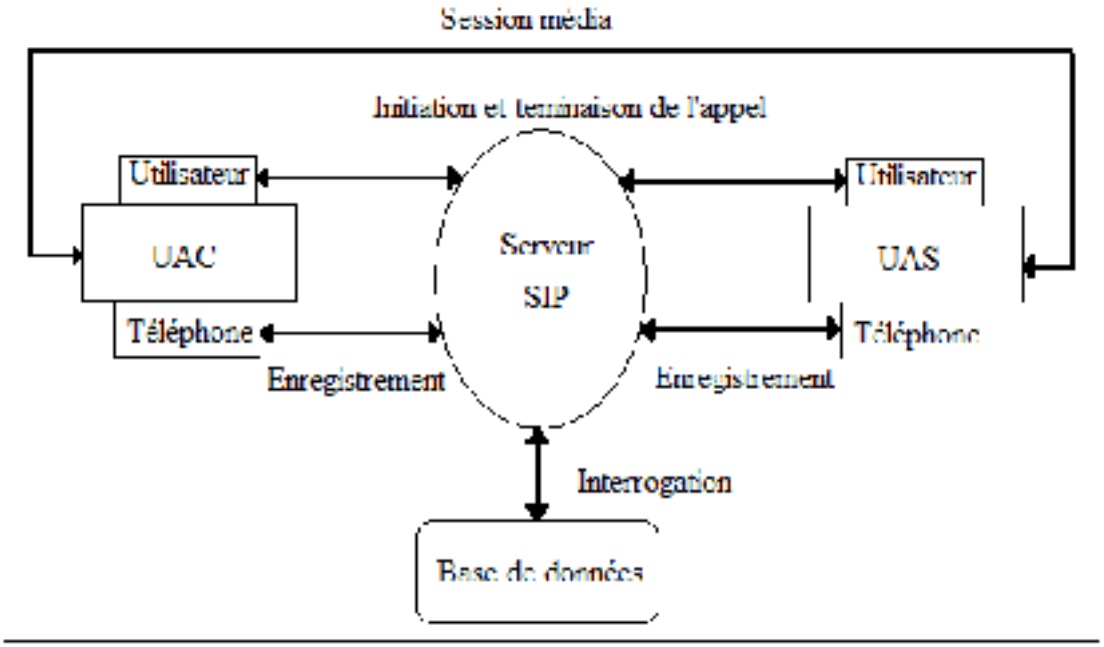

Figure 3.3 Un système standard d'évaluation SIP

Selon la figure 3.3, le serveur SIP joue le rôle d'un serveur Proxy/Registrar capable de créer et terminer les sessions SIP entre deux UAs qui ne connaissent pas leurs emplacements respectifs. Naturellement, l'association URI/Adresse-IP est stockée préalablement dans une base de données. Le Proxy peut donc interroger cette base de données pour diriger les messages vers le destinataire.

Les UA consiste en deux entités principales :

- L'utilisateur est une entité humaine, son rôle est d'initier et terminer les appels;
- Le téléphone est une entité machine : programmé pour initier des opérations d'enregistrement.

Afin de comprendre le fonctionnement de ce système, les quatre modèles suivants sont élaborés : le modèle utilisateur, le modèle enregistrement, le modèle UAC et le modèle UAS. L'élaboration de ces modèles a été inspirée du travail (Erich Nahum, 2007).

3.2.1 Le modèle utilisateur

Le modèle utilisateur est élaboré afin de capturer la façon avec laquelle un usager interagit avec le protocole SIP. Le modèle d'état fini de Markov est utilisé afin de représenter les états, les événements et les transitions. Dans ce modèle, les utilisateurs restent stables dans un état pour une période de temps déterminée. Par la suite ils transitent vers un autre état pendant qu'ils effectuent une action. Les utilisateurs peuvent aussi retourner à leur état précédent.

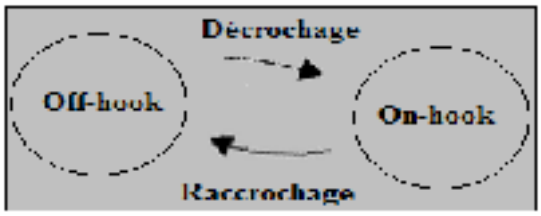

Figure 3.4 Le modèle utilisateur de Markov
Tirée de (Erich Nahum, 2007)

La figure 3.4 représente un Modèle de Markov très simple avec deux états de transition. Quand un événement arrive (i.e. : sonnerie) une action est exécutée tout en effectuant une transition qui peut finir soit par un nouvel état soit par un état précédent.

3.2.2 Le modèle enregistrement

Le comportement du téléphone est modélisé d'une façon périodique et assimile les actions qui sont déclenchées avec et sans l'intervention de l'être humain.

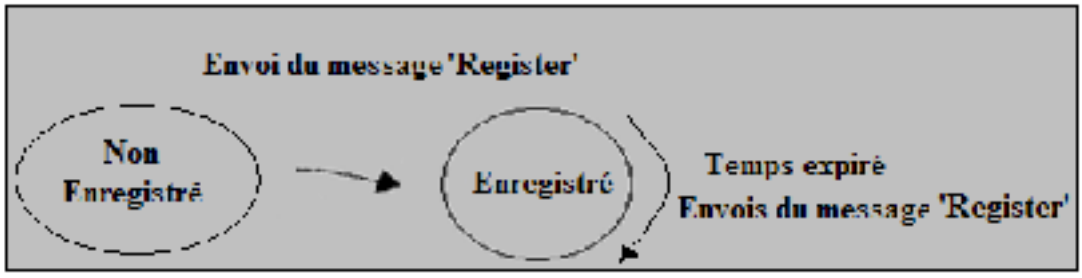

Figure 3.5 Le modèle Enregistrement de Markov
Tirée de (Erich Nahum, 2007)

Pour le cas d'enregistrement, la machine envoie un message 'Register' puis elle se met à l'état '*Enregistrement : en cours*'. Le serveur authentifie le téléphone en lui envoyant un message '401 Proxy Authentication Request'. L'UAC reçoit le message et réponds par les informations d'authentification sans changer son état. Le serveur l'authentifie et envoie un message '200 OK'. L'UAC reçoit le message et transite vers l'état '*Enregistré*'.
Les téléphones font des demandes d'enregistrement périodique, parce que leur durée de validité expire continuellement.

3.2.3 Le modèle UAC

En se basant toujours sur le même principe, un modèle UAC de Markov a été élaboré selon le besoin. Le cycle de transition d'un état à autre commence au début de l'appel et se termine à sa fin. Les événements d'établissement d'appel sont définis par l'envoi des messages

permettant de transiter vers un état menant à établir un appel. Par contre, les événements de rejet sont déclenchés, bien évidemment, par l'envoi des messages '503 Services unavailable'. Un UAC qui reçoit un tel message ne peut envoyer aucune demande au serveur en question (J.Rosenberg, 2002). Dans les architectures 'Serveur-serveur', le UAC peut avoir recours à d'autres serveurs alternatifs. Ce qui n'est pas le cas dans l'architecture étudiée.

La figure 3.6 illustre le modèle utilisé. L'UAC commence par envoyer un message 'Invite' afin d'initier un appel. En suite il se met à l'état '*Initiation*'. Lorsqu'il reçoit le message '100 Trying', il transite à l'état '*Sonnerie : attente*'.

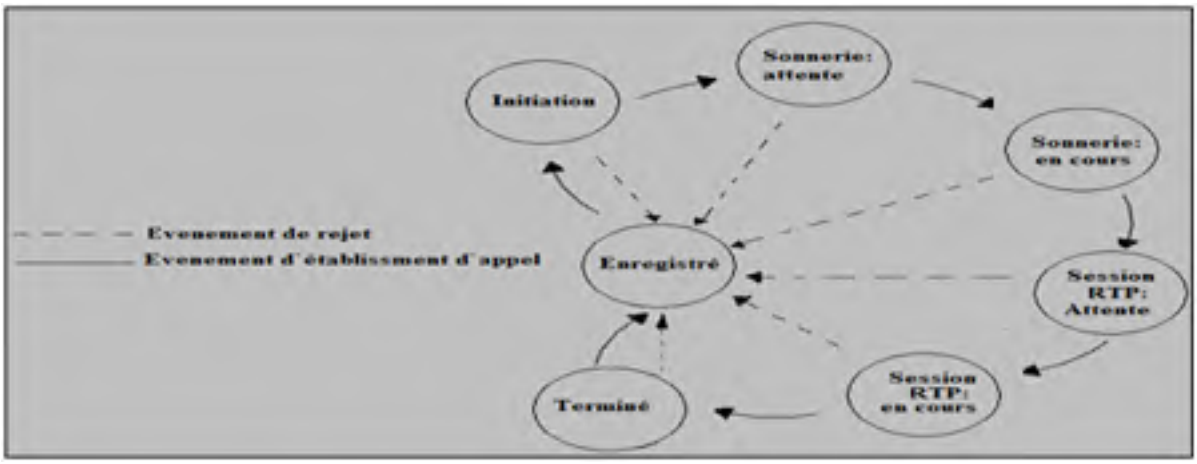

Figure 3.6 Le modèle UAC de Markov

Lorsque le UAC reçoit le message '180 Ring' son état change à '*Sonnerie : en cours'*. Le répondant décroche et un message '200 OK' est transmis. Le UAC reçoit le message et répond par le message 'Ack' tout en se mettant à l'état ' *Session RTP : Attente*', une fois l'UAS reçoit le message 'Ack', la session RTP est établit, ce qui mène l'UAC à changer son état à '*Session RTP : en cours*'.

À la fin de la conversation, un message 'BYE' est envoyé et son état transite vers '*Terminé* '. L'UAS répond par un message '200 OK' et les deux agents se mettent à l'état '*Enregistré* '. Toutefois, un rejet peut être effectué pendant n'importe quel état. Dans ce cas,

le serveur envoie un message ‘503 Service unavailable’ à l’émetteur qui, de son rôle, se met à l’état ‘Enregistré’.

3.2.4 Le modèle UAS

La figure 3.7 montre les différents états et les événements qui déclenchent les transitions. Le UAS reçoit un message ‘Invite’, répond par le message ‘180 Ring’ et change son état à ‘*Sonnerie : en cours*’.

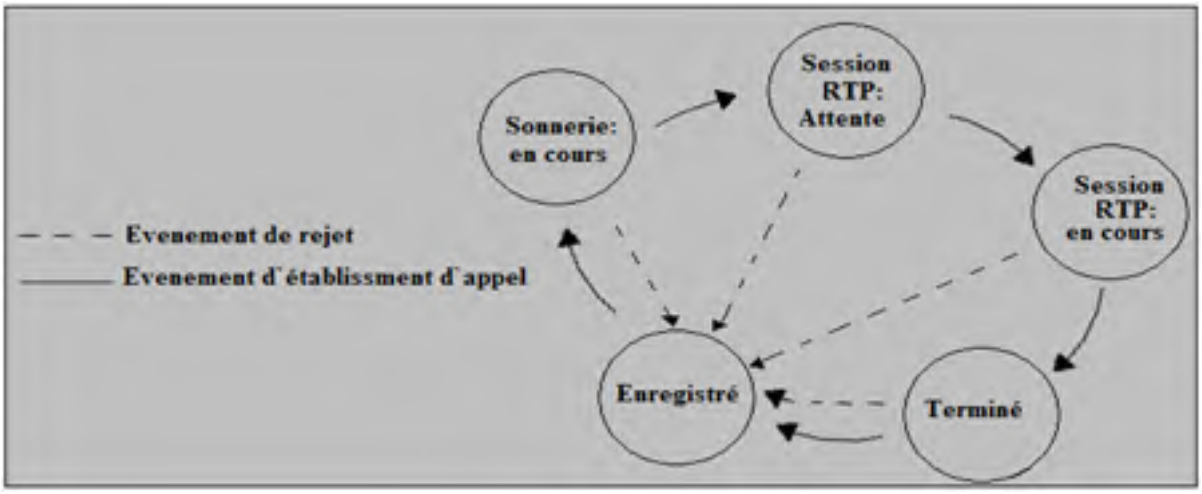

Figure 3.7 Le modèle UAS de Markov

Lorsque le téléphone est décroché, le message ‘200 OK’ est envoyé en changeant l’état à ‘*Session RTP : Attente*’. L’UAS transite vers ‘ *Session RTP : encours* ‘une fois le message ‘Ack’ a été reçu. La terminaison de l’appel suit le même principe décrit dans le modèle UAC.

Contrairement au modèle Enregistrement et utilisateur, les événements qui déclenchent les transitions dans les modèles UAC et UAS peuvent être reliés soit à une intervention humaine (envoi de ‘Invite’ par le UAC) soit un à une réaction d’une machine (réception du message 'Invite' et envoi de ‘180 Ring’ par le UAS). Toujours dans le cas de Alice et Bob, l’envoi du

message ‘Invite’ nécessite l’intervention de Alice, par contre l’envoi du message ‘180 Ring’ ne dépend aucunement de Bob. Dans ce cas, l’intervention est purement machine.

3.2.5 Les délais, les rejets et les modèles utilisés

Toujours dans les mêmes modèles, il existe des délais plus critiques par rapport à d'autres. C’est pour cette raison que, dans ce mémoire, les types de délai et de rejet ont été catégorisés selon trois types principaux : inacceptable, peu acceptable et tolérable. Exceptionnellement à l’état ‘*sonnerie : attente*’ Il existe une relation intime entre les rejets et les délais, un délai est inacceptable implique que le rejet est aussi inacceptable et vice versa. Le tableau 3.1 donne une idée détaillée sur cette relation.

Tableau 3.1 État vs délais et rejets

État	**Inacceptable**	**Peu acceptable**	**Tolérable**
Initiation		Délai, rejet	
Sonnerie: attente	Rejet	Délai	
Sonnerie: en cours	Délai, rejet		
Session RTP: attente	Délai, rejet		
Session RTP: en cours			Délai, rejet
Terminé			Délai, rejet

À titre d’exemple, la durée de l’état ‘*Session RTP : Attente’* est plus sensible par rapport à celle de l’état ‘*Initiation*’. Et donc, le rejet des messages ‘200 OK’ et ‘Ack’ est considéré inacceptable, au contraire au message ‘Invite’, son délai et rejet sont peu acceptable. De ce qui concerne l’exception de l’état ‘*sonnerie : attente*’, tout rejet est considéré inacceptable puisque le serveur a accepté de traiter le message ‘Invite’, et donc le téléphone sonnera dès que l’UAS le reçoit (on admet que le UAC est toujours prêt à prendre les appels). Un rejet du message ‘180 Ring’ signifie que l’appel sera rejeté après la sonnerie du téléphone de l’UAS. Ceci est considéré inacceptable. Par contre, le délai que le UAC passe dans le même

état est considéré peu acceptable parce qu'il existe une certaine tolérance entre la fin de la numérotation et le début de la sonnerie.

De plus, la durée de l'état '*Sonnerie : en cours*' est un délai qui peut être causé par l'utilisateur et qui n'est pas nécessairement dû à la congestion du serveur. Durant la congestion, l'appelant et l'appelé sentent la dégradation de la QoS selon le message qui subit le délai. À titre d'exemple, lorsque l'UAS reçoit le message '180 Ring' le téléphone se met à sonner, l'appelant décroche et un message '200 OK' est émis. Tant que le message 200 OK n'est pas reçu par l'UAC, l'appelant devrait toujours entendre la tonalité de sonnerie, pensant que l'appelé n'a pas encore décroché. La sonnerie s'arrête une fois le message 200 OK est reçu par l'UAC (J.Rosenberg, 2002).

Il est très important de mentionner le délai des sonneries, lorsqu'on parle du délai d'une session SIP, notamment lorsqu'on parle des délais causés par la congestion. Une session qui a duré 4s alors que le téléphone a sonné 3.5s est une situation tout à fait différente que celle qui a duré 4 secondes, mais le téléphone a sonné pendant 1 seconde. Dans cet exemple, le temps de signalisation est respectivement 0.5s et 3s. Ce qui implique une grande différence.

3.3 L'architecture de test proposée

Dans les réseaux en général, il est très important d'assurer une certaine gestion d'utilisation de ressource afin d'avoir une bonne extensibilité. L'idée consiste à éviter la surcharge de certaines ressources tandis que d'autres sont sous-utilisées. Pour se faire, il est courant de placer certains nœuds devant les serveurs d'application pour des fins de sécurité, répartition des charges ou encore de gestion de flux (Jing, Hongliang et Weimin, 2008).

L'architecture proposée se base sur un élément clé situé devant le serveur SIP. Il joue le rôle d'un GFSIP (Gestionnaire de flux SIP) capable d'éliminer les risques de surcharge, ignorer les messages retransmis et garantir les délais selon le besoin. Cette entité assurera la bonne

gestion des messages afin d'éviter les rejets inacceptables et diminuer le taux de rejets des nouveaux appels.

L'architecture devrait simuler un réseau d'entreprise qui se base sur un seul serveur SIP pour établir les communications VOIP. Dans cette architecture, tous les messages doivent passer par le GFSIP et le serveur SIP pour des fins de contrôle et de surveillance. Cela diffère de quelques architectures où le serveur SIP ne reçoit que les messages 'Invite', '180 Ring' et '200 OK', le reste des messages s'échange directement entre les UAs.

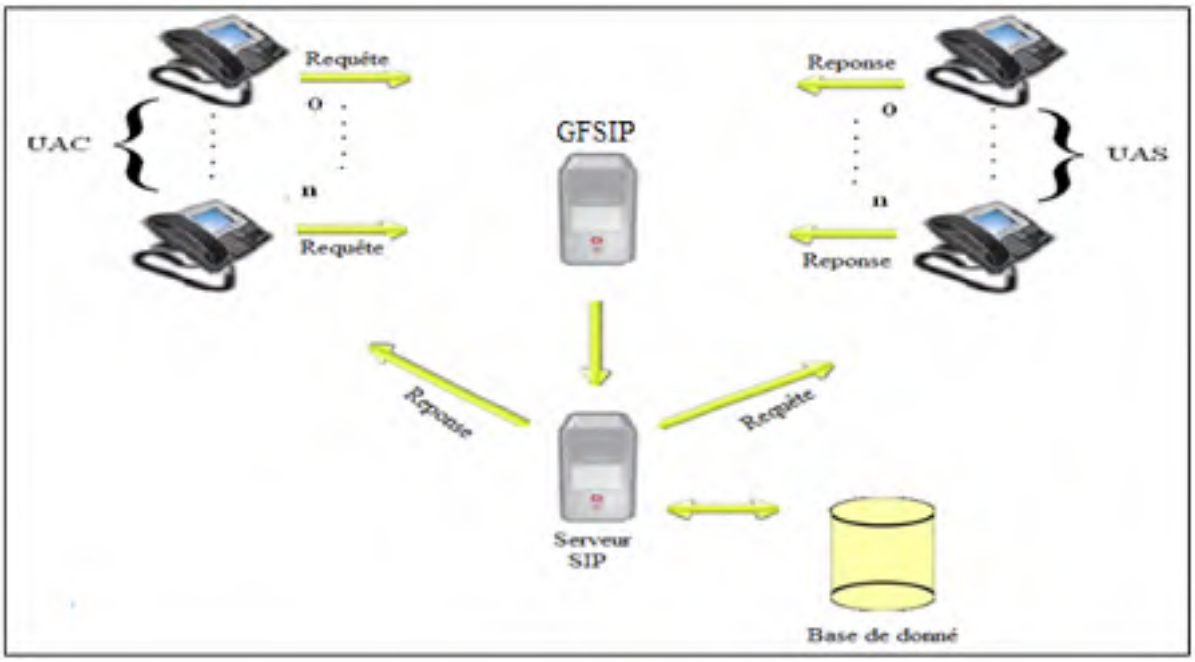

Figure 3.8 L'architecture de test proposée

La figure 3.8 donne une idée de l'architecture en question ainsi que la manière d'échange de messages entre les différentes entités. Pour communiquer, les UAs envoient tous les messages au gestionnaire de flux, car ils ne sont pas au courant de l'architecture du système. Pour eux, le GFSIP est un serveur SIP.

Le GFSIP a pour but d'alléger la charge du serveur SIP. Il reçoit tous les messages provenant des UAs, les classifie dans des files d'attente préétablies et selon la discipline souhaité, et il

les transmet au serveur SIP selon sa capacité de traitement. En cas de rafale de trafic, le GFSIP devrait protéger le serveur SIP contre la surcharge, puisqu'il est la porte d'entrée de tous les messages SIP.

Le serveur SIP est un serveur 'Stateful' qui se caractérise par une file d'attente très courte. La durée d'attente des messages dans les files d'attente du gestionnaire du flux crée un certain lissage de trafic de telle sorte que les messages arrivent selon la vitesse de traitement.

Le serveur SIP est au courant de l'architecture du réseau. C'est la seule entité capable de communiquer avec tous les éléments du réseau SIP (UAs, GFSIP et la base de données). Il traite les messages émis par le gestionnaire du flux et les transmet directement aux UAs sans passer par aucune entité SIP intermédiaire; contrairement aux autres messages qui transitent via le GFSIP.

Lorsque le serveur SIP reçoit un message d'établissement de session, il consulte la base de données afin d'acquérir l'adresse IP du destinataire. La base de données est mise à jour périodiquement à travers le processus d'enregistrement. Ceci est effectué continuellement lorsque les durées de validité sont expirées. La base de données a été élaborée afin d'héberger les informations reliées aux appels ainsi que les statistiques obtenues à la fin du test. Son implémentation sera décrite en détails dans le prochain chapitre.

3.4 L'architecture du gestionnaire du flux SIP proposée

Comme son nom l'indique, le GFSIP a pour but de gérer le flux SIP afin d'aboutir à une qualité de service acceptable. Son efficacité apparaît clairement lors de la période de rafale. Il est doté de quelques fonctionnalités qui sont, naturellement, effectuées par le serveur SIP. Certes, ceci permettra une sorte de répartition de charge. L'architecture générale du GFSIP est illustrée dans la figure 3.9. Cette dernière présente ses modules internes ainsi que l'interaction entre les différentes entités communicantes (UAs, Serveur SIP, GF).

Provenant des UAs, les messages SIP passent préalablement par le module de détection des messages retransmis. S'il s'avère que le message est dupliqué, il sera ignoré, sinon il passe au module suivant. Le protocole SIP utilise UDP comme protocole de transport, donc la fiabilité de la transmission demeure relativement faible. Pour remédier à ce problème, SIP a recours à la retransmission.

Les messages dupliqués ne servent qu'à augmenter les délais dans les files d'attente, notamment lors de la période de rafale. L'implémentation de ce module à l'entrée du système a pour but d'éviter toute charge et tout délai inutile. Ainsi, seulement les messages importants sont mis dans les files d'attentes, ce qui permet d'économiser énormément leur utilisation. En résultat, un pourcentage important des appels sera admis; ce qui diminue le taux de rejet.

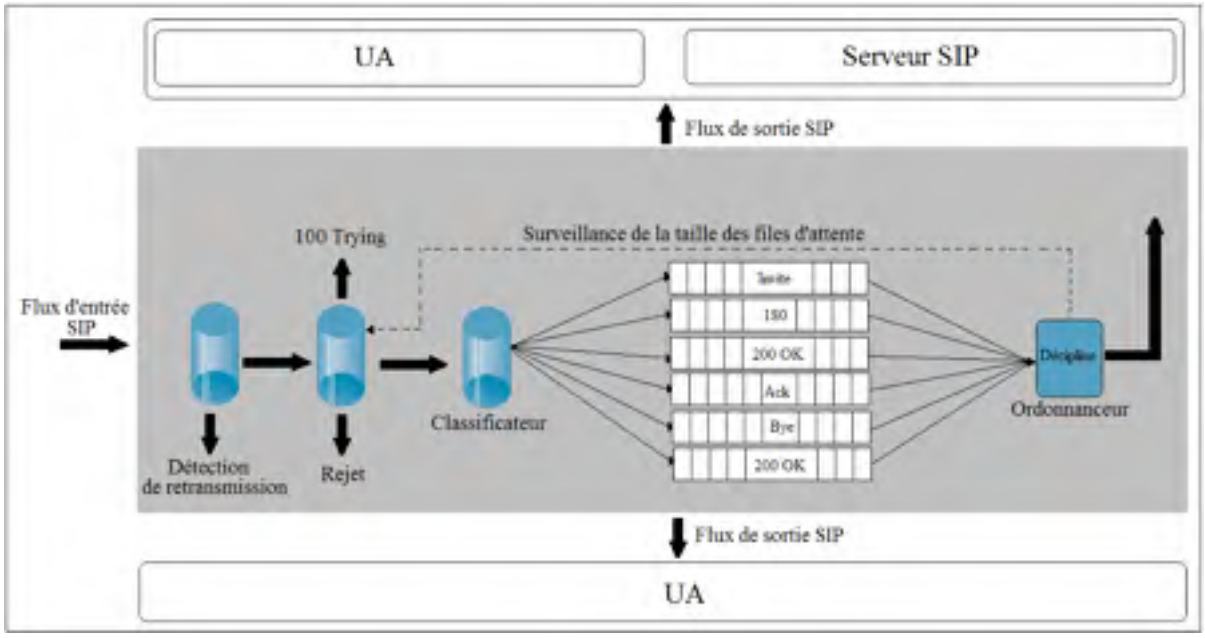

Figure 3.9 L'architecture du GFSIP proposé

Le deuxième module est responsable du rejet des appels en fonction de la taille de la file d'attente des messages 'Invite'. Sa longueur est consultée périodiquement afin de prendre la décision de rejet. La taille maximale de cette file d'attente a une valeur configurable (par l'administrateur) qui dépend du SLA (Service Level Agreement) souhaité. Ainsi, plus la taille est grande plus les délais de réponses deviennent importants. Lorsque la taille de la file

d'attente atteint son maximum, l'opération de rejet est déclenchée. Dans ce cas, le rejet peut être cumulatif et donc tous les types de messages seront rejetés, ou dépendant du type de message, dans ce cas le gestionnaire rejette le type déterminé.

De plus, ce module s'occupe aussi d'envois des messages '100 Trying ' comme réponses aux messages 'Invite' et 'Bye' afin d'éviter la retransmission. Dans le cas du message 'Invite', l'envoi du message '100 Trying' est effectué si la valeur de la taille courante de la file d'attente est inférieure au maximum. Et donc la réception d'un tel message signifie que le message 'Invite' a été admis et qu'un message '180 Ring' devrait arriver bientôt.

L'envoi de '100 Trying ' permet d'éviter les retransmissions inutiles durant la rafale. À titre d'exemple, si le délai d'attente du message 'Invite' est de 1 seconde, chaque message de ce type sera transmis au moins deux fois, car la retransmission s'effectue chaque 500ms selon (J.Rosenberg, 2002). Pour tout autre message, la taille de la file d'attente est consultée afin de décider entre l'admission et le rejet. Cette vérification est effectuée au début afin d'éviter d'empiler des messages qui vont être finalement rejetés. Ceci optimisera l'utilisation des files d'attente, et donc les rejets et les délais seront minimisés.

Le classificateur a pour but de classer les messages selon leurs méthodes pour les requêtes, et selon la combinaison : 'méthode, statut' pour les réponses. En d'autres termes : les méthodes sont classées à l'aide de la fonction *getMethode()* puisque le résultat est unique. Par contre, pour classer les messages '200OK', il faut distinguer entre le '200OK' de 'Invite' et celui de 'Bye'. Ainsi, le résultat obtenu permettra de décider dans quelle file d'attente le message sera stocké.

L'ordonnanceur se charge d'appliquer la discipline souhaitée sur les six files d'attente. Chacune d'elles contient des messages de même type. Elles sont basées sur la discipline premier arrivé premier servi (FIFO). Toutefois, l'ordonnanceur applique trois différentes disciplines sur les différentes files d'attente : *First In First Out*, *Priority Queuing*, *Fair*

Queuing. Les tests effectués devraient conclure concernant les effets des disciplines sur le délai et le rejet d'appel, ce qui fait partie des objectifs de ce projet.

Le but de cette architecture est de combiner tous ces modules afin de minimiser toute sorte de délais durant la rafale. Le fait d'ignorer les messages retransmis (module 1 : détection de retransmission), d'éviter la retransmission des messages 'Invite' (module 2 : envoi du message '100 Tring') et de protéger les files d'attente (rejet du message dès sa réception) aura certainement un effet sur la qualité des appels.

3.5 Conclusion

Le but de ce chapitre est de mettre en évidence les éléments critiques qui contribuent à l'établissement de la qualité de service dans une architecture SIP UA-serveur. Les modèles établis ont pour but de définir les états, les événements et les transitions qui sont effectuées lors d'un appel SIP. Que ce soit dans la même session ou autre, la durée de temps d'un état influe énormément sur les autres. Il devient clair que ces délais influent aussi sur le taux de rejet d'appel.

Pour conclure, le fait d'avoir des délais et des rejets *inacceptables*, *peu acceptables* et *tolérables*, est un fait qui nécessite un système de gestion de message SIP. Celui-ci devrait, bien évidemment, être capable d'éviter tout évènement 'inacceptable', diminuer ce qui est 'peu acceptable' et commencer par le 'tolérable' s'il y a des prix à payer.

CHAPITRE 4

SCÉNARIOS, IMPLÉMENTATIONS ET TECHNOLOGIES UTILISÉES

Le test de performance d'un système SIP nécessite le choix d'un environnement d'exécution fiable. Ainsi, les composants techniques utilisés doivent être dotés d'une forte capacité de précision ainsi que d'une performance dans le cas de la congestion.

Les composants logiciels utilisés dans ce projet ont été utilisés dans plusieurs travaux de recherche tels que (Aziz, Elramly et Ibrahim, 2010; Gao et al., 2009; Van Den Bossche et al., 2006). La performance était satisfaisante, ce qui leur a permis d'avoir une certaine crédibilité. La précision dans un système SIP est un élément essentiel, car l'exécution de ses modules se fait en milliseconde et microseconde. L'analyse de sa performance devrait inclure beaucoup de détails, notamment lorsque les délais sont concernés. Cette partie du rapport commence par l'introduction des outils utilisés dans les tests. Par la suite, une description détaillée de l'implémentation des modules utilisés sera présentée.

4.1 Les Servlets SIP de Java

La Servlet SIP est un ensemble d'API (JSR 116 (Oracle, 2011b) et JSR 289 (Oracle, 2011c)) de JAVA qui définit des extensions de haut niveau pour les serveurs SIP. Elle permet à l'application SIP d'être déployée et gérée selon un modèle basé sur la Servlet. elle se considérée comme une technologie robuste qui renfonce la communication multimédia que le protocole SIP établit. La Servlet SIP fournit un accès abstrait aux opérations de signalisation pour un déploiement rapide des services de nouvelle génération {Chris Boulton, 2009 #20}

La Servlet SIP est dotée d'un conteneur qui permet la réception des requêtes et les réponses selon les spécifications de (J.Rosenberg, 2002). Sa conception facilite au programmeur l'implémentation des applications et évite la concentration sur les opérations reliées au protocole SIP. L'API en question fournit une panoplie de classes, Interfaces et méthodes permettant de manipuler les messages reçus selon le besoin. Le conteneur SIP est une entité

configurable permettant au développeur de changer ses paramètres pour l'adapter à son application. Ce changement peut aussi être effectué au niveau des spécifications du RFC 6132, ce qui donne au programmeur une certaine liberté d'implémentation.

La Servlet SIP est installée dans les machines contenant le code du serveur SIP et du GFSIP. Elle fournit tous les outils nécessaires permettant de relier les messages interchangeables entre les différents UAs. Pour plus d'information sur ces outils, il est recommandé d'avoir recours à (Chris Boulton, 2009; Oracle, 2011b; 2011c).

4.2 Bref aperçu sur SIPp

SIPp (Hewlett-Packard, 2004) est un outil de génération de trafic libre pour le protocole SIP. Son comportement est contrôlé par des scénarios XML, dans lesquels des modèles machines et utilisateurs sont définis. Il est capable d'établir des centaines d'appels par seconde selon des scénarios préétablis (selon le besoin) afin d'évaluer la performance du système sous test. Il supporte une panoplie de technologies telles que (Hewlett-Packard, 2004) :

- protocole de transport: TCP, UDP;
- authentification avec SSL (TLS);
- retransmissions UDP;
- les expressions régulières;
- RTP replay and echo;
- variable, arithmétique, comparaison.

SIPp est un outil programmable (langage C), il a été développé par *Hewlett-Packard* et a été mis à la disposition de la communauté SIP pour tester plusieurs systèmes tels que : les serveurs mandataires, B2BUAs, Serveur média SIP, SIP PBX, etc… Il a été utilisé dans plusieurs travaux de recherche pour des fins d'évaluations de performance. Outre, plusieurs communautés de recherche, notamment IBM, ont contribué à son développement. Ils sont parvenus à corriger une multitude d'erreurs en ajoutant plusieurs fonctionnalités qui ont rendu l'outil convenable pour l'évaluation.

SIPp reporte les résultats (statistiques) obtenus selon un intervalle prédéterminé en utilisant des fichiers de statistiques. Les temporisateurs et les compteurs peuvent être définis dans le code XML afin d'avoir les délais et les statistiques souhaités. Il se dote de plusieurs commandes qui sont exécutables au début et durant le test.

PCAP (Sourceforge, 2010) (Packet Capture) est une libraire qui permet de générer un trafic RTP vers une destination. Il peut être jumelé avec SIPp pour présenter un système VOIP complet. Ceci peut se faire à travers les commandes *PCAP play* en utilisant les attributs *play_pcap_audio* et *play_pcap_video* (Hewlett-Packard, 2004) . Le trafic RTP peut être capturé et enregistré par des outils d'analyse de protocole réseau tel que : *Wireshark, TCPDump*. PCAP supporte plusieurs types de trafic temps réel tel que: trafic voix, vidéo, voix et vidéo, etc.

La version de SIPp utilisée dans les tests est 3.2. Elle est installée dans les machines qui jouent le rôle de l'appelant et l'appelé. Pour une meilleure performance, 'Linux' est le système d'exploitation utilisé. SIP est disponible sous forme d'un code source qui doit être compilé avant d'être utilisé. La compilation nécessite les prés-requis suivants (Hewlett-Packard, 2004) :

- compilateur C++;
- l'une des librairies 'curses' ou 'ncurses' (Free Software Foundation, 2011);
- pour l'authentification et le support de TLS : OpenSSL;
- pour le support de trafic RTP: libpcap et libnet;
- pour les pauses distribuées: la librairie GSL (Free Software Foundation, 2011).

Pour compiler SIPp , les commandes suivantes sont exécutées :

```
gunzip sipp-3.2.tar.gz
tar -xvf sipp-3.2.tar
cd sipp-3.2
make ossl
```

Lorsque SIPp est installé, il est possible des faire des tests basiques entre un UAC et UAS en utilisant des scénarios intégrés. Pour ce faire, les deux commandes suivantes sont exécutées :

```
./sipp -sn uas                (Cette commandes exécute SIPp avec le scénario UAC intégré )
./sipp -sn uac 127.0.0.1      (Cette commandes exécute SIPp avec le scénario UAS intégré)
```

L'adresse IP 127.0.0.1 signifie que les deux instances s'exécuteront dans la même machine. L'attribut 'sn' désigne que le scénario utilisé est un scénario intégré.

SIPp supporte une panoplie de commandes qui permet tous les types de tests avec les statistiques adéquates. Il donne accès à une session de commande ou il est possible d'insérer des commandes durant le test tel que : 'set rate 50'. Cette commande permet d'augmenter le débit d'appel à 50 appels par seconde. Pour plus d'information il est recommandé de consulter (Hewlett-Packard, 2004).

4.3 Les scénarios utilisés

Les scénarios consistent en l'échange des messages entre les entités SIP afin d'établir les appels. L'intégration d'un GFSIP dans le réseau SIP a permis d'obtenir des scénarios différents de ceux utilisés dans un réseau SIP standard. L'échange de paquets de signalisation s'effectue entre quatre entités (UAC, UAS, GFSIP, Serveur SIP). Un serveur AAA (Authentication, Authorization and Accounting) est mis en place pour des fins de sécurité et

de surveillance. La méthode d'authentification implémentée est très basique. Son but est de simuler l'authentification lorsqu'un GFSIP est utilisé.

L'authentification concerne seulement les messages 'Invite' , 'Bye' et 'Register'. Le but est de vérifier l'identité des UAs qui initient les appels, afin de s'assurer que seuls les membres autorisés peuvent effectuer les appels. La vérification des messages 'Register' permettra d'éviter l'enregistrement des appareils non autorisés. La vérification des messages 'Bye' évitera les attaques malicieuses ayant l'intention de terminer des appels déjà en cours.

La méthode de sécurité utilisée n'est pas capable d'immuniser le système contre toutes les attaques. Elle est utilisée afin de simuler les scénarios proposés avec l'authentification. De plus, la sécurité du système en question n'est pas le but de ce projet, toutefois, elle peut faire partie d'un travail futur.

4.3.1 Le flux de paquet de signalisation et les délais associés

La figure 5.1 détaille la session SIP qui a été implémentée. L'UAC initie l'appel par le message 'Invite' qui est envoyé directement au GFSIP. Ce dernier répond par un message '100 Trying' et transmet le message 'Invite' au serveur SIP. Ceci vérifie l'en-tête afin de récupérer les informations d'authentification. S'il s'avère que ces informations sont manquantes, il enregistre le message 'Invite' reçu et envoie un message '407' afin d'avertir le UAC de l'authentification. L'UAC répond par un message 'Ack' contenant un nom d'utilisateur et un mot de passe.

La base de données joue le rôle d'un serveur AAA. Elle est consultée afin de vérifier l'état d'enregistrement de l'UAC, ainsi que pour récupérer l'adresse IP du destinataire pour lui transmettre le message 'Invite' et les messages subséquents. Lorsque le serveur SIP reçoit un message 'Ack' avec les informations d'authentification, il consulte la base de données afin de vérifier la validité des informations. Juste après, le message 'Invite' est transmis à l'UAS concerné.

L'échange des messages d'authentification (Ack et 407) s'effectue directement entre les UAs et le serveur SIP sans passer par le GFSIP. On considère que ces messages font partie du traitement du message 'Invite', et donc, ils doivent être traités tout de suite. En conséquence, le temps de traitement du message 'Invite' sera plus élevé par rapport aux autres messages, ce qui affectera le délai 'Invite-180'.

L'UAS reçoit le message 'Invite' et répond par un message '180 Ring'. Ce message passe par le GFSIP qui le transmet au serveur SIP, qui de son rôle, l'envoie à l'UAS (Voir la figure 4.1).

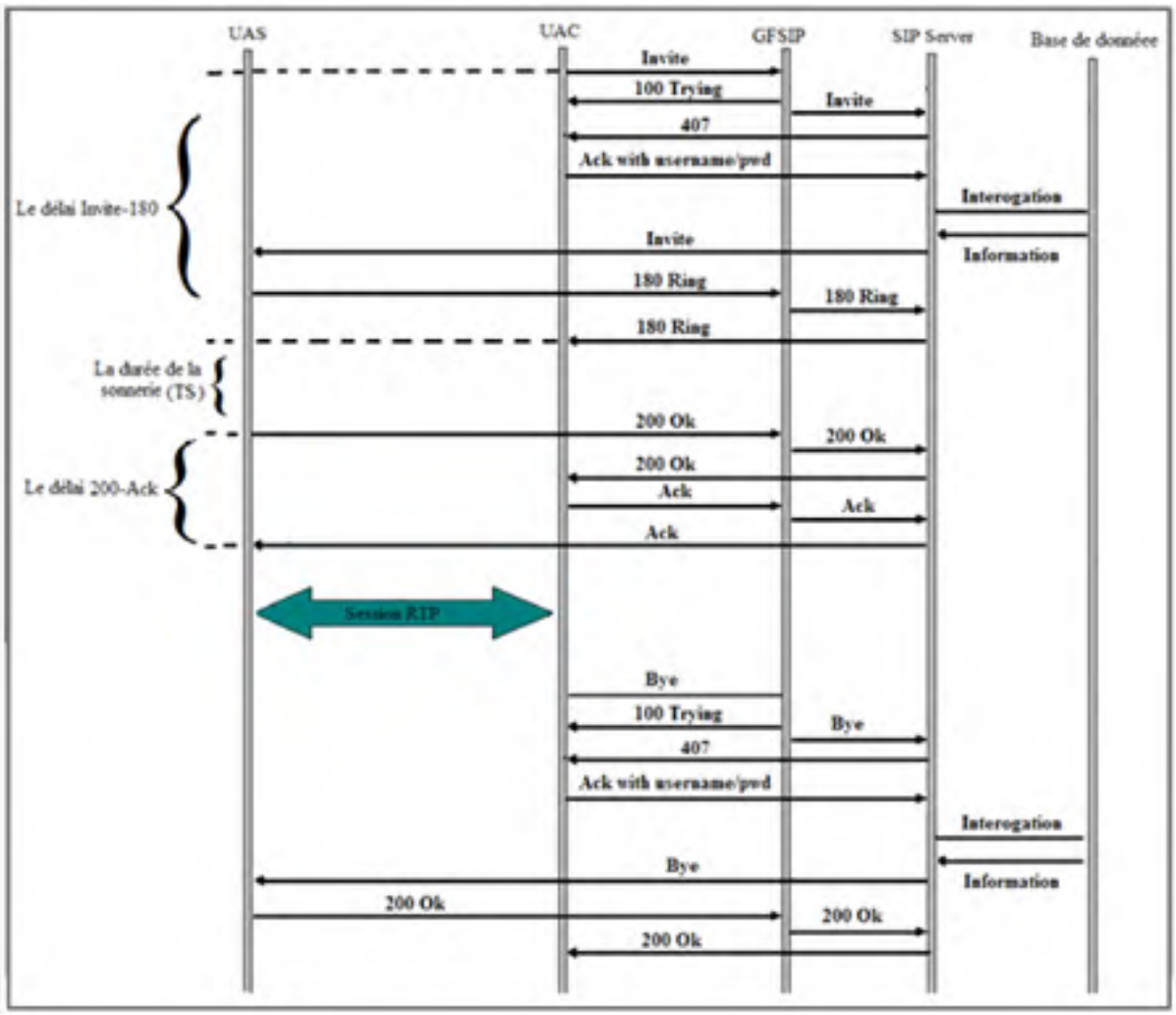

Figure 4.1 Le scenario proposé d'un appel VOIP

La durée entre l'envoi du message 'Invite' (UAC) et la réception du message '180 Ring' (UAS) est intitulée le délai 'Invite-180'. Ceci est considéré comme un délai '*peu acceptable*' puisqu' il existe une certaine tolérance chez les usagers. Comme c'est déjà mentionné dans le chapitre précédent, un appelant peut avoir une certaine tolérance au délai entre la fin de la numérotation et le début de la sonnerie. La tolérance dont on fait allusion dépend de quelques secondes. Et donc, profiter de cette tolérance peut avoir un grand avantage pendant la rafale de trafic.

Le délai 'Invite-180' (intitulé Temps de Réponse ou 'Response Time' dans d'autres travaux de recherche) comprend deux délais ayant des tolérances différentes. D'une part, le délai du message 'Invite' qui est la durée entre l'envoi du message par l'UAC et sa réception par l'UAS. Et d'autre part, le délai du message '180 Ring' qui est la durée entre l'envoi du message par le UAS et sa réception par le UAC.

Le délai du message 'Invite' est considéré *peu acceptable*, car l'UAS n'est pas encore notifié de l'appel. À ce point, l'attente est seulement du côté de l'appelant. De plus, l'appelant a toujours une certaine tolérance de délai avant qu'il entende la tonalité de la sonnerie. Lorsque le UAS reçoit le 'Invite', il répond par un message '180 Ring' le moment où le téléphone se met à sonner. Durant cette étape, les deux côtés sont au courant de l'appel.

Afin d'assurer l'intégrité de la session SIP, l'appelant devrait entendre la tonalité de sonnerie le plus rapidement possible, ainsi, le délai du message '180 Ring' devient *inacceptable*. En résultat : seul le délai du message 'Invite' sera ciblé afin de profiter de la tolérance du délai 'Invite-180'.

Le délai entre la réception du message '180 Ring' (par l'UAC) et l'envoi du message '200 OK' (par l'UAS), est la durée de sonnerie du téléphone. Sauf dans le cas du retard des messages, un grand pourcentage de ce délai dépend de l'appelé.

Le délai ‘200-Ack’ est la durée entre l’envoi du message ‘200 Ok (Invite)’ et la réception du message ‘Ack’ par le UAS. Ce délai comprend deux délais inacceptables : le délai du message ‘200 OK’ et celui du message ‘Ack’. Il est clair qu’il n’existe aucune tolérance au délai lorsque l’appelé décroche le téléphone. Ceci dit que l’échange des paquets de voix doit être effectué le plus rapidement possible dès que l’UAS reçoit le message ‘Ack’. Bien évidemment, tout délai menaçant la transmission de ces deux messages devrait être éliminé.

Le message 'Bye' est envoyé par le UAC afin de terminer l'appel, le GFSIP le reçoit, répond par un message ‘100 Trying’ et le transmet au serveur SIP. Ceci authentifie l’UAC suivant la même procédure utilisée pour le message 'Invite'. Lorsque le UAS reçoit le message 'Bye', il répond par un message ‘200 OK (Bye)’ qui termine la session SIP courante. Le délai entre ces deux messages est considéré tolérable. Le fait de profiter de cette tolérance aura sûrement un grand avantage sur la QoS.

Cette étude considère que le prix à payer durant la rafale doit commencer par les messages à basses priorités sans influencer les sessions qui sont déjà en cours. L’implémentation des files d’attente dans GFSIP a pour but d’assurer une certaine gestion de traitement des messages selon leurs natures, ce qui facilitera l’implémentation des priorités.

4.3.2 Le flux de paquet d’enregistrement

Le processus d’enregistrement et d’authentification est implémenté afin de simuler les opérations effectuées par un serveur SIP.

Une instance SIPp indépendante a été implémentée afin de générer périodiquement des messages ‘Registrer’. La base de données contient le temps d’expiration pour chaque téléphone.

La figure 4.2 représente le processus d’enregistrement. L’UAC envoie un message ‘Registrer’ au serveur SIP. Ce dernier répond par une demande d’authentification (Message

‘401’). L’UAC renvoie le même message avec un en-tête ‘*Authorization*’ qui contient un nom

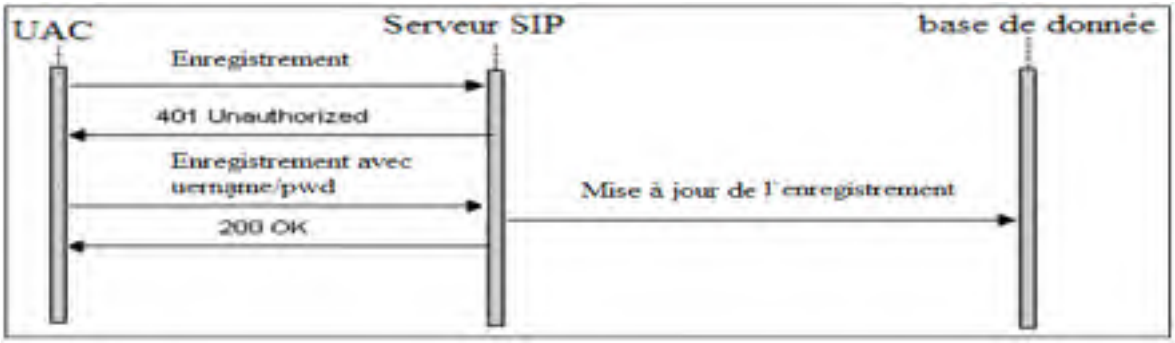

Figure 4.2 Le processus d’enregistrement

d’utilisateur et un mot de passe pour des fins d’authentification. À l’aide de la base de données, Le serveur l’authentifie et met la base de données à jour (mis à jour du temps d’expiration). Un message ‘200 OK’ est envoyé pour confirmer l’enregistrement.

Les scénarios vus jusqu’à date présentent une idée théorique sur l’échange de messages entre les différents UAs. Dans la partie qui suit, l’accent est mis sur l’implémentation de ces scénarios en utilisant un code XML.

4.4 Phase de l’implémentation

Cette partie du rapport détaillera l’implémentation des différentes entités menant à un système SIP opérationnel. Une partie du code élaboré sera expliquée afin de mettre en évidence l’approche proposée. La syntaxe des codes XML utilisés dans ce projet se trouve dans la source (Hewlett-Packard, 2004).

4.4.1 L’implémentation des UAs

Trois types d’implémentation seront introduits : l’instance SIPp Client-Appel, l’instance SIPp Client-Enregistrement et l’instance SIPp Serveur.

4.4.1.1 L'instance SIPp Client-Appel

L'instance SIPp client (Appel) est le code XML utilisé permettant au SIPp d'initier les appels selon le scénario mentionné précédemment. Ce code est exécuté dépendamment du débit d'appel. Seulement les parties importantes du code seront discutées.

L'initiation de l'appel est déclenchée par le code suivant :

```
 INVITE sip:[field1]@etsmtl.ca SIP/2.0
 Via:SIP/2.0/[transport][local_ip]:[local_port];branch=[branch]-
 [call_number]
 From: [field0] <sip:[field0]@etsmtl.ca>;tag=[call_number]
 To: [field1] <sip:[field1]@etsmtl.ca>
 Call-ID: [call_id]
 CSeq: 1 INVITE
 Contact: <sip:[field0]@etsmtl.ca>
 .
 .
 .
</send>
 <!— état actuel: Initiation… -->
```

L'instruction 'send' définit l'opération d'envoi du message 'Invite' chaque 500ms (`retrans="500"`) tant que le message '100 Trying' n'a pas été reçu. Le 'start_rtd' déclenche le temporisateur numéro '1' qui sert à calculer le délai 'Invite-180'. L'attribut 'CDDATA' définit le contenu du message 'Invite'. Lorsque ce dernier est émis, le UAC se mets à l'état initiation et attend le message '100 Trying'. Le code qui permet sa réception est le suivant :

```
<recv response="100" optional="true">
</recv>
<!— état actuel: Sonnerie : attente … -->
```

La réception du message '100 Trying' signifie que le GFSIP a accepté le traitement du message 'Invite' et que le message '180 Ring' arrive sous peu. Durant cette période l'UAC est à l'état '*Sonnerie : attente*'. Le serveur SIP reçoit le message 'Invite' et authentifie

l'UAC à travers les deux messages '407' et 'Ack'. Lorsque le message '180 Ring' est reçu, l'UAC change son état à '*Sonnerie : en cours*'. Le code qui permet cette opération est :

```
<recv response="180" optional="true" rtd="1">
</recv>
<!— état actuel: Sonnerie : en cours … -->
```

L'instruction rtd="1" arrête le temporisateur numéro '1' qui a été déclenché lors de l'envoi du message 'Invite'. Ceci permettra d'obtenir, pour chaque appel, le délai 'Invite-180'. Du côté de l'UAS, il existe un code qui permet de simuler la durée de la sonnerie. Le message '200 OK' est donc envoyé après une période de temps prédéfinie. Ainsi, lorsque l'appelé décroche, le message '200 OK' est reçu par le UAC à travers le code :

```
<recv response="200"  >
</recv>
<!— état actuel: session RTP : attente … -->
```

L'UAC répond par un message 'Ack' :

```
<send>
 <![CDATA[
 ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
 Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
 From: Appelant <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
 .
 .
 .
 ]]>
 </send>
<!— état actuel: session RTP : attente … -->
<!— état actuel: session RTP : en cours … (UAS reçoit le 'Ack') -->
```

Dès que le UAS reçoit le message 'Ack', la session RTP commence. La période d'échange de paquet de voix est simulée par une pause. Durant cette période, aucune opération de signalisation n'est effectuée.

```
<pause milliseconds = "3000" />
```

Ce code doit être implémenté dans l'UA qui initie la terminaison de l'appel (l'envoi du message 'Bye'). En général, ce type de code est toujours implémenté dans l'instance SIPp

qui envoie le premier message après la fin de la pause, car il n'y a pas moyen de notifier les autres instances de l'existence d'une pause durant une étape de signalisation. À la fin de la conversation, le message 'Bye' est envoyé à l'aide du code suivant :

```
<send >
 <![CDATA[
 BYE  sip:[service]@[remote_ip]:[remote_port] SIP/2.0
 Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
 .
 .
 .
 ]]>
 </send>
<! — État actuel: session RTP : terminaison -->
```

Just après, le message '100 Trying' est reçu. Par la suite le message 'Bye' est authentifié. le message '200 OK' est reçu pour terminer la session en cours :

```
<recv response="200" crlf="true" >
</recv>
<! — État actuel: Enregistré … -->
```

4.4.1.2 L'instance SIPp Client-Enregistrement

L'instance SIPp client (Enregistrement) est le code XML élaboré permettant au SIPp d'effectuer l'enregistrement et l'authentification selon le scénario mentionné précédemment. Seulement les parties importantes du code seront discutées.

Contrairement à l'instance SIPp client-Appel qui dépend du débit d'appel, l'instance SIPp Client-Enregistrement est sous forme d'une boucle qui envoie continuellement les messages 'Register'.

```
<pause min="0" max="1200000" />
<label id="5" />
<send>
<![CDATA[
REGISTER sip:etsmtl.ca SIP/2.0
Via:SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
From:sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
.
.
.
]]>
</send>
```

Le code XML de cette instance commence par une pause aléatoire entre 0 et 1200 secondes. Une étiquette définie par l'identificateur '5' détermine le début de la boucle. Le champ *'Expires'* définit la durée de vie de l'enregistrement en seconde. Cette valeur sera prise en considération par le serveur SIP afin de déterminer l'état d'enregistrement du téléphone. En tout temps le serveur SIP répond par une demande d'authentification :

```
<recv response="401" auth="true">
</recv>
```

Un fichier de type CVS a été créé afin de contenir les noms d'utilisateur et mots de passe. Son nom est mentionné lors de l'exclusion de l'instance SIPp client-Enregistrement. Le fichier doit être structuré sous forme de champs de données afin de permettre au SIPp de chercher l'information. Le même type de fichier est utilisé par l'UAC pour inclure les informations de l'authentification.

Le deuxième message 'Regsiter' doit contenir le nom d'utilisateur et le mot de passe. Son code est comme suit :

```
<send>
<![CDATA[
REGISTER sip: etsmtl.ca SIP/2.0
Via:SIP/2.0/[transport][local_ip]:[local_port];branch=[branch]
-[call_number]
From:"[field0]"<sip:[field0]@etsmtl.ca>;tag=[call_number]
To: "[field0]"<sip:[field0]@etsmtl.ca>
[field2]
Call-ID:[call_id]
CSeq: [cseq]  REGISTER
Contact: "[field0]" <sip:[field0]@192.168.10.47>
Expires: 300000
Max-Forwards: 65
]]>
</send>
```

Les mots clés [field0] et [field1] désignent le premier et le deuxième champ du fichier CVS qui contiennent respectivement le nom d'utilisateur et le mot de passe. Ceci permettra au serveur SIP de vérifier les informations et envoyer un message de confirmation à travers le code suivant :

```
<<recv response="200">
</recv>
<pause next="5" milliseconds="1200000" />
```

Finalement, une autre pause d'une durée de 2000 secondes est déclenchée. L'attribut *next="5"* permet de revenir à l'étiquette '5' afin de reprendre la boucle.

4.4.1.3 L'instance SIPp serveur

Le code de l'instance SIPp serveur (UAS) se base sur le même principe que celui du client-Appel (UAC). Le code suit le scénario UAS mentionné auparavant. Il commence par la réception du message 'Invite' et se termine par l'envoi du '200 OK'. Cette implémentation considère que seul l'UAC initie la terminaison d'appel et donc, les UAS reçoivent le message 'Bye' durant toute la période du test.

4.4.2 L'implémentation du Serveur SIP

Le serveur SIP est sous forme d'une Servlet SIP de Java qui implémente les APIs définis dans les spécifications JSR 116. Il a été élaboré afin de couvrir toutes les fonctionnalités requises pour un serveur Proxy/Registrar de VOIP. Il implémente les opérations d'enregistrement et récupération des informations reliées aux utilisateurs à l'aide d'une base de données. Son code a été élaboré à l'aide de l'outil de développement Netbeans (Oracle, 2011d). Cet outil se dote de plusieurs fonctionnalités permettant de faciliter la tâche aux programmeurs. Notamment la correction des erreurs de syntaxe et les outils de débogage.

Pour faire fonctionner le serveur, une panoplie d'outils doit être préalablement installée :

- Netbeans IDE 6.9 et les plugins SIP nécessaires (Optionnel);
- Java SE Development Kit;
- Glassfish Comunication Server;
- moteur de base de données (Derby).

Le serveur SIP écoute sur le port 5060 et reçoit les messages provenant du GFSIP. Il se dote de deux méthodes principales permettant de différencier entre les requêtes et les réponses reçus: *dorequest()* et *doresponse()*. En général le code des applications serveur d'un système SIP dépend largement du besoin du programmeur. Et donc le délai d'exécution diffère d'une implémentation à autre. Les APIs offrent une multitude de classes qui facilitent l'implémentation de toute sorte de modules.

Cette implémentation repose essentiellement sur deux axes principaux :

- Élaboration d'un système d'enregistrement (Registrar);
- Élaboration de la fonctionnalité de liaison entre UAC et UAS (Proxy).

Les deux fonctionnalités présentent le comportement d'un serveur SIP standard afin de tester l'approche proposée sur un système SIP réel. Le code du serveur SIP est de quelques centaines de lignes. Seules les parties importantes du code seront expliquées.

4.4.2.1 Le proxy

Les requêtes et réponses SIP sont échangées entre les différentes entités SIP à travers le 'Proxying'. Essentiellement, le serveur SIP devrait être capable de différencier entre les requêtes initiales et celles subséquentes. Dans cette implémentation une requête initiale est une requête dont le proxy ne possède pas une session déjà établie. Dans ce cas, une nouvelle session doit être créée afin de reconnaître tous les messages subséquents qui lui appartiennent. Les APIs utilisés se dotent d'une class 'Proxy' capable de faire ce travail. Ceci consiste à créer un Object de cette classe lors de la réception du message 'Invite' comme suit :

```
protected void doInvite(SipServletRequest req) {
  .
  .
  .

Proxy proxy = req.getProxy();
p.setRecordRoute(true);
p.proxy(req.getRequestURI());
  .
  .
  .
}
```

Dans cet exemple, l'objet proxy aura la fonction de relier les messages subséquents qui appartiennent à la même session SIP. Toutefois, il fallait tenir en considération l'architecture proposée et faire passer tous les messages par le GFSIP avant le serveur SIP. Pour résoudre le problème, une nouvelle implémentation du proxy a été effectuée afin d'impliquer le GFSIP dans l'échange du flux SIP.

La Servlet, en tant que tel, n'est pas capable de gérer les dialogues des différentes sessions. Toutefois, les APIs utilisés offrent un mécanisme qui permet de différencier entre les sessions SIP à travers deux interfaces essentielles : *SipSession* et *SipApplicationSession*.

La figure 4.3 montre la relation entre ces deux éléments (Oracle, 2011a). Un objet de l'interface *'SipApplicationSession*' peut contenir plusieurs objets '*SipSession'*. Dans chacun des deux, il est possible de créer des attributs qui servent essentiellement à enregistrer les données spécifiques pour l'objet courant. Ces données jouent un rôle très important dans la récupération des sessions SIP, ce qui permet de diriger les messages vers la session à laquelle ils appartiennent.

Ce modèle a été élaboré afin de faciliter la convergence des applications, notamment le protocole HTTP et SIP. Ainsi, une seule instance d'application peut contenir plusieurs sessions point-à-point qui utilisent différents protocoles (Oracle, 2011a).

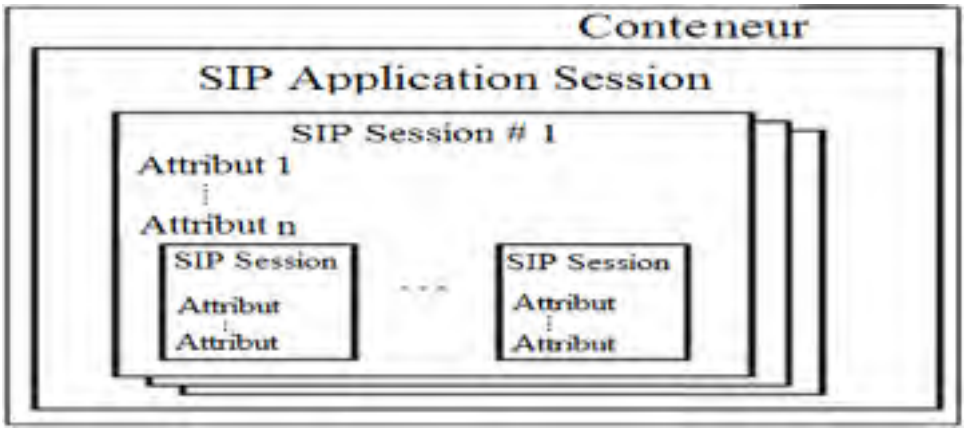

Figure 4.3 SipSession vs SipApplicationSession
Tirée de (Oracle, 2011a)

Pour posséder le contrôle de tous les messages que la Servlet reçoit, un traitement particulier des messages a été effectué. Ce traitement sera expliqué en bref afin d'avoir une idée sur la fonction du proxy implémenté.

Lorsque le serveur SIP authentifie le message 'Invite', la servlet crée un nouvel objet de la classe '*sipservletRequest*' qui appartiendra à une nouvelle session '*SIP session*'. Cet objet sera affecté à la requête 'Invite' reçue selon le code suivant :

```
NewInvite = sipFactory.createRequest(InviteReceived, true);
```

Cette instruction permet de créer une requête SIP '*NewInvite'* avec la majorité des en-têtes incluses dans le message reçu '`InviteReceived`'. Par la suite, deux attributs '*LINK*' sont séparément créés et affectés aux sessions SIP des objets `NewInvite` et `InviteReceived` suivant le code :

```
SipSession InviteReceivedSession = InviteReceived.getSession();
SipSession NewInviteSession = NewInvite.getSession();
leg1.setAttribute(LINK, NewInviteSession);
leg2.setAttribute(LINK, InviteReceivedSession);
NewInvite.setAttribute(LINK, InviteReceived);
```

Selon le code ci-dessus, chaque session contient un attribut qui fait référence à la seconde session. Le message `NewInvite` est celui qui sera transmis au UAS, le serveur SIP sera en mesure de traiter le message '*180 Ring*' reçu grâce à la dernière instruction. Celle-ci lui permettra de récupérer le message 'Invite' `InviteReceived` qui sera utilisé pour créer une réponse '*180 Ring'*. Ceci dit que le message '180 Ring' reçu ne sera pas transféré, mais un nouveau message du même type sera, localement, créé, puis transféré.

Lorsque le message '200 OK' est reçu '*ResponseReceived*', le serveur a recours à un attribut pour garder une copie du message dans sa session SIP. Par la suite, le message 'Invite' `InviteReceived` est récupéré à l'aide de l'instruction *Response Received.getRequest()* qui est utilisée pour transférer le message '200 OK' à l'UAC.

À la réception du message 'Ack', sa session SIP est récupérée afin d'obtenir celle du message `NewInvite`. Ceci a été sauvegardé lors de la réception du message 'Invite' à travers le code mentionné précédemment. La session SIP est utilisée pour récupérer la réponse '200 OK' qui a été sauvegardée et qui sera utilisée pour créer un message 'Ack' à travers le code suivant : `Ack = ResponseReceived.createAck()`.

Finalement, lorsque le serveur SIP authentifie le message BYE '*ByeReceived*', ill commence par la récupération de la session SIP du message `InviteReceived` à travers l'attribut '*LINK'*. Un nouveau message 'Bye' '*NewByeMessage*' est créé à travers la session SIP récupérée. Une copie du message '*ByeReceived*' est sauvegardée dans un attribut appartenant au message '*NewByeMessage*'. Ce dernier est envoyé à l'UAS qui, de son rôle, répond par un message '200 OK'. Ce message est utilisé par le serveur SIP pour récupérer le message '*NewByeMessage*'. Ceci contient un attribut qui fait référence au message *ByeReceived* qui sera utilisé pour créer une réponse '200 OK'.

Cette implémentation traite les messages spécifiés par le RFC 6132 qui sont échangés lors d'établissement et la terminaison d'un appel. Le code des méthodes implémentées se trouve dans l'annexe 1. La partie qui suit traite l'implémentation de l'enregistrement des téléphones indépendamment de l'appel.

4.4.2.2 L'enregistrement

Le serveur SIP reçoit périodiquement des messages 'Register' des différents téléphones pour des fins d'enregistrement. La procédure d'authentification utilisée est très basique. Pour chaque message, le serveur SIP vérifie la valeur de l'en-tête '*Authorization*'. Si elle est égale à 'nul', un message '401 Unauthorized' est émis :

```
statusCode = SipServletResponse.SC_UNAUTHORIZED;
SipServletResponse R401U = InviteReceived.createResponse(statusCode);
R401U.send();
```

Sinon, il compare le nom d'utilisateur et le mot de passe reçu à ceux qu'il a dans la base de données. Par la suite, le l'en-tête '`Expires`' est consulté afin de calculer le temps d'expiration :

```
ExpirationT = system.currentTimeMillis() + (Expiration * 1000);
```

Le serveur dégage les informations comprises dans les en-têtes *FROM* et *CONTACT* et crée un objet de la classe '*Database*' qui les sauvegarde dans la base de données. Ainsi pour chaque utilisateur, les informations suivantes sont enregistrées : le URI de l'en-tête *FROM*, URI de l'en-tête *CONCTACT* et le temps d'expiration de l'enregistrement. De cette manière, il sera possible d'enregistrer les utilisateurs qui se connectent aux différents téléphones. Ceci permettra au serveur SIP de localiser l'UAS lors de la réception d'un message 'Invite'. À la fin de cette opération, un message '200 OK' est créé pour être envoyé à l'UA concerné.

4.4.3 L'implémentation du GFSIP proposé

Le GFSIP est une Servlet SIP installée dans une machine séparée, qui est logiquement situé devant le serveur SIP et qui écoute sur le port 5090. Elle communique, essentiellement, avec les UAC et le serveur SIP. Lors de traitement des messages, le serveur SIP intègre l'adresse IP du GFSIP pour que ce dernier les reçoive en premier, sinon les UAs lui envoient les réponses directement.

La figure 4.4 présente l'algorithme implémenté dans le GFSIP. Les messages passent préalablement par le module de détection de la retransmission. Dans cette implémentation, un appel donné peut être identifié par le champ 'CALLID', car sa valeur demeure unique pendant toute la période du test.

Selon (J.Rosenberg, 2002), seulement les messages 'Invite, 200K (Invite) et Bye' sont retransmis. Le message 'Invite' est retransmis chaque T1=500 ms tant qu'il n'a pas reçu le message '100 Trying' et jusqu'à un maximum de T1*64 secondes. Tandis que le message '200 OK' et 'Bye' sont retransmis chaque T1=500 ms tant qu'ils n'ont pas reçu respectivement les messages 'Ack' et '200 OK'. Dans ce cas, la retransmission dure un maximum de T2=4 secondes.

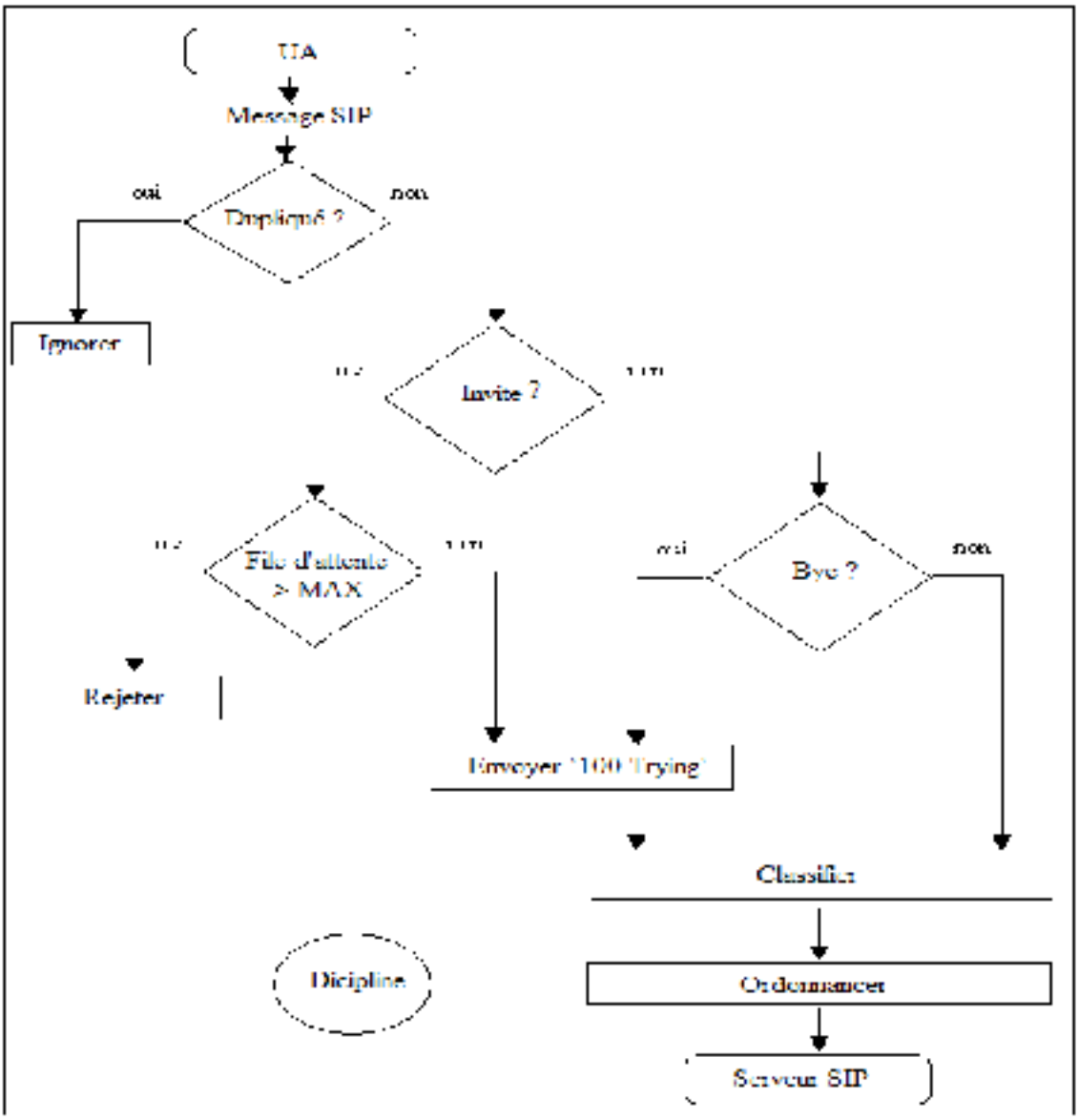

Figure 4.4 L'algorithme implémenté dans le GFSIP

Lorsque le GFSIP reçoit une requête, il vérifie si son *CALLID* existe dans sa base de données. Si c'est le cas, le message est ignoré, sinon, sa méthode/status est sauvegardée. La base de données des messages reçus est sous forme d'une structure de donnée HashMap qui prend deux attributs : identificateur de l'appel 'CallId' et nombre des messages reçu en d'autres termes, le nombre de message reçu devrait s'incrémenter lorsque l'un des trois messages est reçu jusqu' à un maximum de trois :

```
RequestReceived= InviteMessage
callId = InviteMessage. getCallId()
if (HashMap.contains(callId) )
Drop(InviteMessage)
Else
HashMap.add(callId,1)
```

Le test *if(HashMap.contains(callId)* ne s'applique que pour les messages non-Invite subséquents. car le 'CallID' a été déjà sauvegardé à la réception du message 'Invite' et donc, on se retrouve avec le code suivant :

```
ResponseRecieved = 200OkMessage
CallId = 200OkMessage.getCallId()
If (HashMap.contains(callId,3))
Drop(ResponseRecieved)
else
HashMap.modify(callId,3)
```

S'il s'avère que le message n'est pas dupliqué, il passe au module suivant, celui-ci surveille continuellement la longueur de la file d'attente des messages 'Invite' (Le code de la classe des files d'attente se trouve dans l'annexe 2). Si la taille est supérieure ou égale au maximum, un message '503 Service Unavailable' est créé puis transmis à l'UA concerné, sinon le message sera admis. L'admission d'un message Invite nécessite la création d'un message '100 Trying' qui est transmis à l'UA concerné. Ainsi la retransmission des messages 'Invite' sera évitée. Pour les autres messages, ils passent directement au classificateur qui applique la fonction getMethode() sur les requêtes et getMethode() + getStatus() sur les réponses afin de les acheminer aux files d'attente appropriées. Le code du classificateur se trouve dans l'annexe 3.

L'ordonnanceur est sous forme d'un thread de type *Daemon* qui surveille les files d'attente et retire les messages selon la discipline désirée. À titre d'exemple, dans le cas d'un Fair Queuing, le Thread exécute les cinq fonctions suivantes à tour de rôle (Round-Robin), le code complet du FQ se trouve dans l'annexe 4:

```
while (true) {

ProcessQueueInvite();
ProcessQueue180();
ProcessQueue200();
ProcessQueueAck();
ProcessByeQueue();
ProcessBye200Queue();
}
```

Chacune des fonctions se contente de retirer le message qui est au début de la file et de l'envoyer au serveur SIP. Le Thread crée une sorte de boucle qui exécute le contenu de chaque fonction une seule fois dans chaque itération. L'exécution continue tant que la longueur des files d'attente est différente de zéro. Les messages qui arrivent doivent attendre leurs tours, ce qui crée les délais d'attente problématiques qui seront analysés pendant la phase des tests et résultats (prochain chapitre).

4.5 L'architecture de l'environnement expérimental

Le système de test élaboré simule un réseau SIP capable de lier des centaines d'utilisateurs qui communiquent à travers la voix sur IP. On assume que les appels s'effectuent entre les membres du groupe. Chaque usager est doté d'une identité SIP unique permettant à un autre de le joindre. Les éléments : serveur SIP, GFSIP, UA et base de données sont hébergées par des serveurs Linux séparés, et sont reliés à travers un réseau local de 100 Mb/s. L'architecture de l'environnement est comme suit :

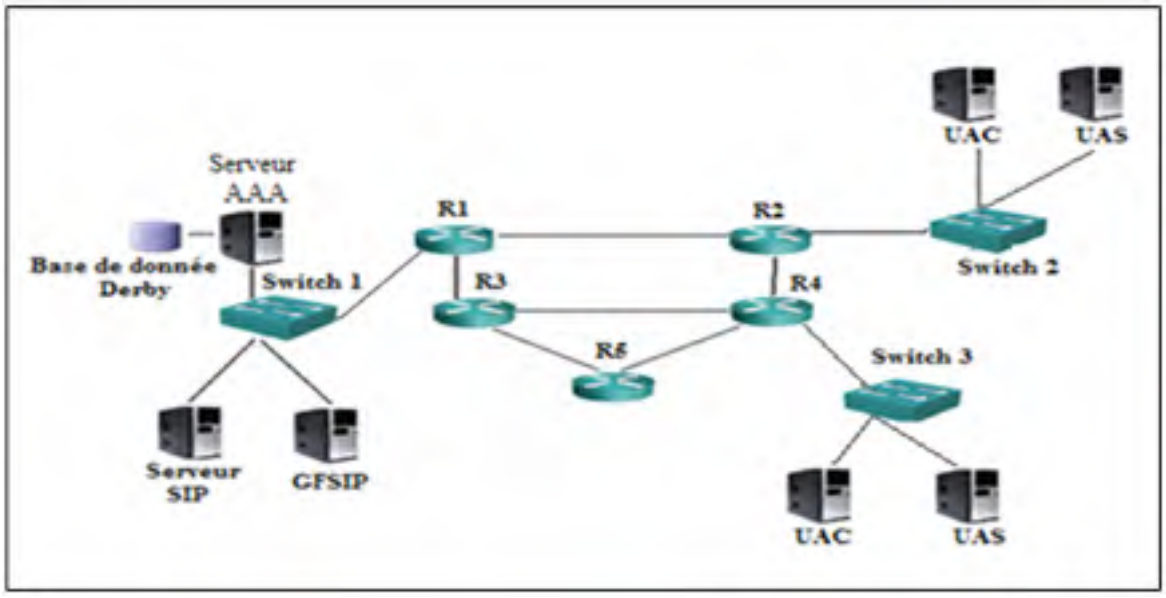

Figure 4.5 L'architecture de l'environnement expérimental

La figure 4.5 illustre l'architecture du réseau Ethernet utilisé. Elle se compose de quelques serveurs Linux reliés entre eux à travers cinq routeurs et trois commutateurs. Les routeurs sont configurés afin de relier les différents réseaux sans tenir en considération la QoS. Les messages seront acheminés selon l'adresse IP de destination d'une façon égale. Les serveurs Linux sont reliés à travers les routeurs et les commutateurs. Ils hébergent les modules développés qui subiront le test de performance. La configuration des serveurs est la suivante :

- Processeur : 3.4 GHz;
- RAM : 2 Go;
- Mémoire cache : 1 Go;
- Systéme d'exploitation: Ubuntu version 10.10.

Le but est d'effectuer les tests sur une architecture semblable à celle d'une entreprise qui se base sur un seul serveur SIP pour relier les téléphones VOIP. En cas de rafale, l'étude se concentra sur le délai et le rejet causé par la surcharge du GFSIP.

Deux machines hébergent les générateurs de trafic UAC, et deux autres hébergent les générateurs UAS. Les appels sont effectués entre UAs de réseaux différents. Dans chaque machine contenant un UA, une instance SIPp d'enregistrement est installée afin de générer les messages d'enregistrement. Ces messages sont destiné directement au serveur SIP, car l'implémentation du GFSIP ne les tient pas en considération, ce qui est parmi les limitations de ce projet.

4.6 Le déploiement

Le système de test élaboré nécessite l'exécution des composants suivants :

- Une servlet SIP déployée dans le serveur d'application Glassfish (Serveur SIP);
- Une servlet SIP déployée dans le serveur d'application Glassfish (GFSIP);
- Une instance de l'outil SIPp (Client-Enregistrement) qui envoie les messages 'Register' au servlet (Serveur SIP);
- Une instance de l'outil SIPp (Client-Appel) qui envoie les messages Invite/Ack/Bye au GFSIP;
- Une instance de l'outil SIPp (Serveur) qui répond aux messages Invite/Bye;
- Création des tables dans la base donnée.

Dans les machines des UAs, un fichier contenant le nom d'utilisateur et mot de passe des utilisateurs doit exister dans le même répertoire que le fichier binaire SIPp (fichier SIPp exécutable) et le fichier Register.xml (le scénario d'enregistrement). SIPp consultera ce fichier pour inclure les informations d'authentification dans le message 'Register'.

Toujours dans la même machine, l'instance SIPp (Client-Enregistrement) est exécutée à l'aide de la commande suivante:

./sipp ServeurSIP_IP_address -sf register.xml -inf nom_du_fichier –i Local_ IP -p 5070

Dans les deux machines ou l'instance SIPp (Client-Appel) existe, la commande suivante est exécutée :

. /sipp -sf uac.xml –rsa GFSIP_IP_address : Port UAS_IP_Address :Port –p Local_Port –i Local_ IP_Address -inf nom_du_fichier –r 10

Les champs FROM et TO de l'en-tête SIP incluront respectivement *Local_ IP_Address et UAS_IP_Address :Port*. L'attribut *'rsa'* avise *SIPp* que les messages générés passeront par un intermédiaire qui est naturellement un serveur SIP. Dans ce cas, c'est le GFSIP.

Dans les machines où l'instance SIPp (Serveur) existe, la commande suivante est exécutée :
./sipp -sf uas.xml -i Local_ IP_Address -p Local_Port

Pour déployer les Servlets implémentées, il faut tout d'abord créer un domaine et démarrer le serveur GCS (Glassfish Comunication Server) à l'aide de la commande :

./asadmin start-domain Nom_du_domain

Par la suite, le code du Servlet est déployé à travers la commande:

deploy SIP_Server_Servlet.war

La base de données (*Derby*) est installée dans une machine Linux séparée qui joue le rôle d'un serveur AAA (figure 4.5). Elle contient trois tables permettant d'enregistrer les différents types d'information durant la période du test. D'une part, ils sont utilisés pour sauvegarder l'information d'enregistrement des utilisateurs, d'autre part, pour sauvegarder les informations reliées à l'appel et l'authentification.

Voici les champs de la table qui contient les informations reliées à l'appel :

- Appelant : VARCHAR(64) NOT NULL
- Appelé : VARCHAR (64) NOT NULL
- Début d'appel : TIMESTAMP NOT NULL
- Durée d'appel : BIGINT NOT NULL

Voici les champs de la table qui contient les informations d'authentification :

- Nom utilisateur : VARCHAR(64) NOT NULL
- Mot de passe : VARCHAR(64) NOT NULL

Voici les champs de la table qui contient les informations d'enregistrement :

- Nom utilisateur : VARCHAR(64) NOT NULL
- AddresseIP : VARCHAR (64) NOT NULL
- Temps d'expiration: TIMESTAMP NOT NULL

Il est recommandé de démarrer la base de données et de créer les tables en premier, par la suite démarrer le GCS et déployer les Servlets. Ensuite, exécuter les commandes de génération de trafic.

4.7 Limitation

Les scénarios élaborés ne prennent pas en considération les erreurs. On assume que tous les utilisateurs existent parmi le groupe et sont toujours disponibles à prendre les appels et qu'il n'existe pas des erreurs d'enregistrement ni de non-disponibilité ou possibilité de Hacking. L'UAC est toujours celui qui raccroche en premier, et l'échange des messages d'enregistrement s'effectue directement avec le serveur SIP.

4.8 Conclusion

Les chapitres 3 et 4 ont donné une idée détaillée sur la conception et l'implémentation d'un système de test automatisé, capable de simuler un environnement VOIP. Les outils utilisés sont présentés ainsi que quelques parties du code élaboré. Outre, les Servlets se dotent de

plusieurs compteurs et temporisateurs pour des fins de statistique. Les tests visent, essentiellement, la période de rafale ou les délais et le nombre d'appels rejetés augmentent lorsque les files d'attente se remplissent.

Le GFSIP est la porte d'entrée de tous les messages destinés au serveur SIP. En conséquence, ce dernier sera protégé contre la congestion puisque la rafale se dirigera vers le GFSIP en premier. Ainsi, le comportement du GFSIP est considéré critique durant cette période. Le chapitre qui suit, aura pour but d'effectuer des tests sur le système élaboré afin d'analyser le comportement du GFSIP versus délai et rejet. La conclusion devrait s'effectuer sur l'effet de la gestion des messages sur les éléments problématiques (délai et rejet) durant la rafale.

CHAPITRE 5

TESTS ET RÉSULTATS

Les tests seront effectués en respectant les scénarios définis dans les chapitres précédents. Le GFSIP applique trois disciplines sur les six files d'attente établies. Pour chacune d'elle, les délais et le nombre d'appels rejetés seront analysés.

Les disciplines appliquées sur les files d'attente permettent un contrôle sur les messages qui seront transmis et ceux qui doivent attendre. Ainsi, ils ont un effet très important sur les délais de transmission des paquets. En conséquence, plus le délai augmente, plus le taux de rejet est important. La décision prise sur les messages à envoyer influe énormément sur les éléments problématiques mentionnés précédemment, notamment lorsque les messages ont des tolérances différentes. Le but est de comparer les résultats des différentes disciplines pour savoir laquelle est convenable aux tolérances définies auparavant.

5.1 La dépendance entre messages SIP

La dépendance entre les messages SIP est apparue d'une façon très claire lors de l'application des différentes disciplines. Il a été remarqué que le débit de tous les messages subséquents dépend du débit du message 'Invite'. La figure 5.1 illustre cette dépendance sous forme d'un diagramme.

Le débit du message '180 Ring' dépend explicitement et complètement du débit du message 'Invite'. Sauf en cas de perte de paquet, l'augmentation du débit de 'Invite' implique l'augmentation instantanée du débit du message '180 Ring' (on admet que l'appelé est toujours prêt à prendre l'appel). Par contre, aucun des autres messages n'en dépend (message '180 Ring') parce que sa perte n'influe pas sur la continuité du processus de signalisation. En d'autres termes, l'appel peut être établi même si ce message est perdu. Au pire des cas, l'appelant commencera la conversation sans entendre la tonalité de la sonnerie.

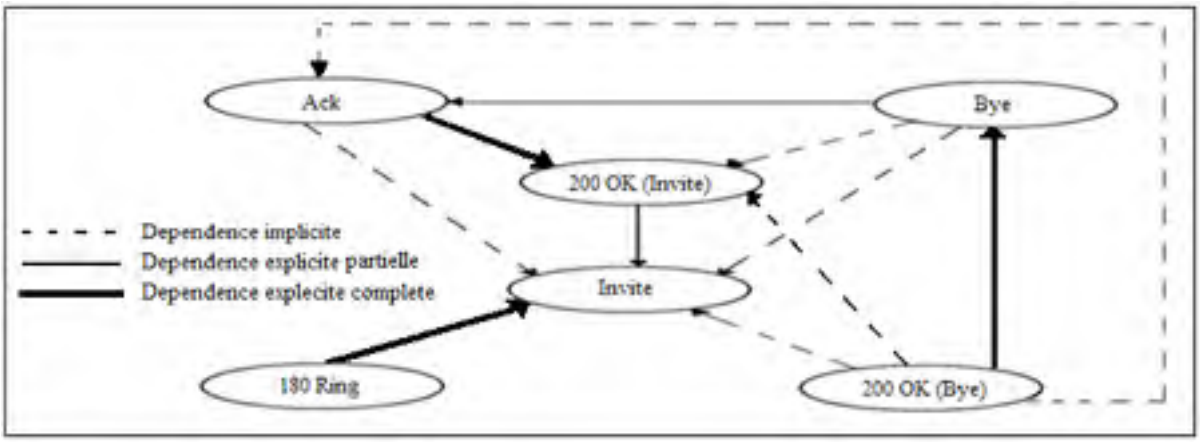

Figure 5.1 Le diagramme de dépendance entre messages SIP

Le débit du message ‘200 OK (Invite)’ dépend explicitement et partiellement du message ‘Invite’. Naturellement, l’augmentation du débit de ce dernier ne veut pas dire que le débit du message ‘200 OK (Invite)’ sera élevé. Parce que son débit dépend aussi de la distribution du temps de la sonnerie. C’est pour cette raison que la dépendance est considérée partielle.

Le message ‘Ack’ dépend explicitement est complètement du message ‘200 OK (Invite)’, c'est-à-dire que si le GFSIP reçoit une rafale des ‘200 OK (Invite)’, il recevra automatiquement une rafale du message ‘Ack’, ce qui est aussi le cas pour les messages ‘Invite’ et ‘180 Ring’. Toutefois, il dépend implicitement du débit du message ‘Invite’ parce que son débit est principalement contrôlé par le message ‘200 OK (Invite) ‘, sans oublier que la présence des messages ‘Ack’ dépend aussi des messages ‘Invite’.

Le débit du message ’Bye’ n’a aucune dépendance explicite complète. Il dépend implicitement du message ‘Invite’ et ‘200 OK (Invite)’ et explicitement du message ‘Ack’. Ce dernier ne tient pas un contrôle total sur le débit du message ‘Bye’ parce que la distribution de la durée de la conversation joue un rôle très important sur le lissage du trafic ’Bye’.

Suivant le même principe, il est facile de déduire que le message ‘200 Ok (Bye)’ a une dépendance explicite et complète au message ‘Bye’, et dépend implicitement des autres messages.

Pour conclure, la dépendance implicite est une dépendance où le débit d’un message dépend du débit d’un ou plusieurs messages intermédiaires, tandis que dans une dépendance explicite partielle, il n’y a aucun message intermédiaire. Seule la distribution du temps entre les deux messages est le facteur influant sur le débit. Dans une dépendance explicite complète, comme son nom l’indique, le débit du dépendant dépend complètement du débit du message précédent. Toutes ces dépendances seront concrétisées par les résultats obtenus.

5.2 Tests et statistiques

Les débits générés durant les tests ont pour but de simuler la nature du trafic voix dans la réalité. Ils débutent par un débit léger de 10 appels/s où le GFSIP a toutes les ressources pour traiter les messages avec des délais négligeables. Le débit augmente d’une façon brusque en moins d’une seconde. Il atteint une valeur maximale prédéterminée et y reste pour une période de temps bien précise. À la fin de cette période, le débit revient à sa valeur initiale pendant une certaine période de temps.

Les tests effectués durent une période de 30 minutes, dans laquelle neuf rafales à différents débits ont été générées. Chacune d’elle dure 100 secondes. Pour chaque appel, le temps de sonnerie du téléphone est fixé à 500ms. La conversation dure 3s. Plusieurs débits de rafale sont générés afin d’évaluer la performance du système implémenté. Les délais obtenus sont calculés selon une moyenne obtenue chaque seconde. La longueur des files d’attente est une valeur obtenue au début de chaque seconde. Un compteur de rejet est mis en place afin de compter le nombre d’appels qui ont été rejetés pendant chaque période de rafale.

Les analyses visent principalement la période de rafale. Les files d’attente, les délais et les rejets sont les éléments cruciaux qui subissent des variations importantes nécessitant une

concentration bien pointue. Toutefois, d'autres éléments ayant une grande influence sur le système sont apparus, notamment ce qu'on appelle le ramasse-miettes.

5.3 L'effet du ramasse-miettes (Garbadge Collector)

La gestion de mémoire dans JAVA se fait d'une manière automatique à travers le ramasse-miettes (Garbage Collector où GC*)*. Contrairement à d'autres langages tels que C/C++, le programmeur a la possibilité de réserver et libérer les ressources mémoires manuellement. Les algorithmes du GC implémentés dans le JVM (Java Virtual Machine) ont posé quelques problèmes de latence dans les systèmes VOIP. Sa configuration dépendant intimement du code implémenté, parce que le nombre d'objets créés durant le 'Runtime' dépend de l'implémentation.

Dans un environnement Java, le programmeur n'a pas de contrôle sur l'exécution du GC lors de l'exécution du programme. Il peut proposer son exécution dans des endroits du code, mais cette exécution n'est pas garantie. L'utilisation d'un GC permet d'éviter les erreurs d'allocation de mémoire commises par les programmeurs. Ceci est considéré comme l'un des avantages d'un GC. Toutefois, certains problèmes ont été relevés concernant les applications sensibles au délai tel que la voix sur IP.

Le problème majeur des codes VOIP dans Java est la latence. Cette dernière apparaît clairement lors des arrêts successifs causés par le GC. En effet, son exécution arrête complètement l'exécution du programme; ce qui engendre des délais qui sont parfois inacceptables.

La configuration d'un GC repose sur plusieurs attributs que le programmeur doit configurer dépendamment de l'implémentation. Le choix de la configuration peut avoir comme objectif de minimisation de temps d'exécution, dans ce cas l'élimination des pauses causées par le GC n'est pas si importante. Comme il peut aussi viser l'optimisation de la latence, dans ce cas les pauses doivent être minimisées. (Van Den Bossche et al., 2006).

Plusieurs algorithmes GC existent, et donc certains travaux de recherche ont été effectués afin d'étudier l'effet de ces algorithmes sur les délais. Ce projet implémente la méthode utilisé dans (Van Den Bossche et al., 2006) qui c'est basée sur la concaténation de deux algorithmes, '*Concurent collector*' et le '*Parallel collector*'.

La Figure 5.2 présente une comparaison entre deux politiques de GC différentes : *Concurent + Parallel* ' et le GC par défaut (*Optthruput)*.

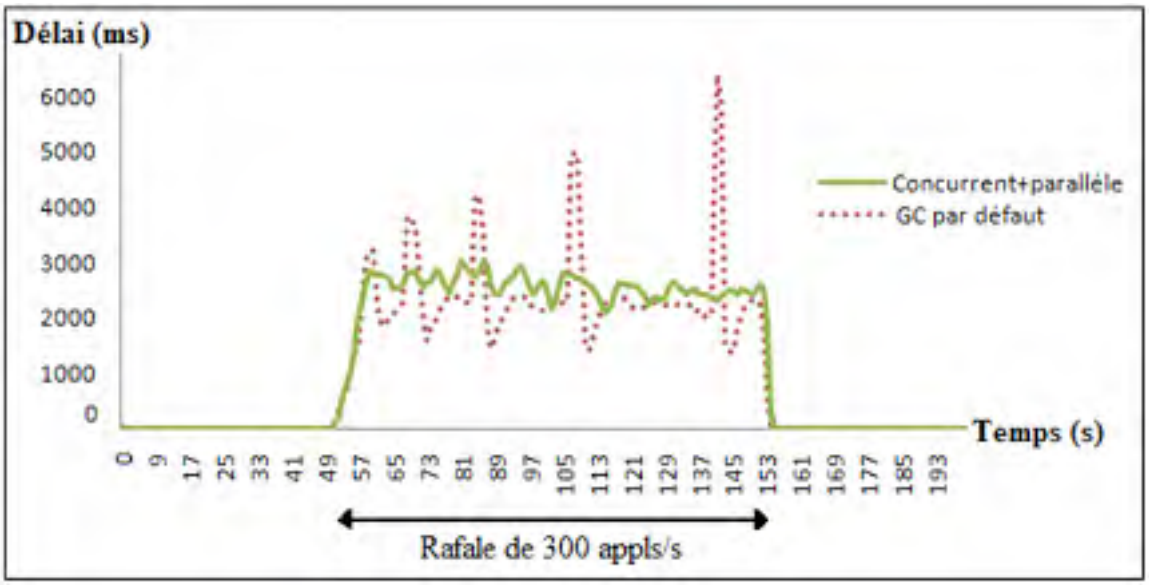

Figure 5.2 L'effet des algorithmes du GC sur le délai

Les oscillations remarquées dans le graphe sont essentiellement dues à l'arrêt instantané du GC par défaut. Car son exécution cause un arrêt du JVM, ce qui engendre une augmentation instantanée des délais. En ce qui concerne le deuxième algorithme (*Concurrent+parallèle*), les délais suivent une variation relativement fiable. L'algorithme '*Parallel*' déclenche une multitude de Thread qui accélère l'exécution des tâches et minimise les pauses, au moment où l'algorithme '*Concurrent'* s'exécute en parallèle avec l'application. La concaténation de ces deux algorithmes a évité les 'Sauts' de délais imprévus et a permis aux délais d'avoir des variations imperceptibles par l'usager durant toute la période de rafale.

L'utilisation de l'algorithme adéquat ne garantit pas des meilleures performances. Le GC se dote de quelques paramètres qui doivent aussi être configurés convenablement. La mémoire utilisée par le JVM (*Heap*) est parmi les paramètres essentiels à configurer. Sa valeur devrait être établie soigneusement. Elle ne doit pas être très grande, car dans ce cas les pauses du GC seront plus grandes, et elle ne doit pas être trop petite, car le JVM risque d'avoir des problèmes d'insuffisance de mémoire. Le *Heap* se compose de trois sections majeures intitulées : Génération (*Young Generation* (YG)*, Tenured Generation* (TG), *Permanent Generation (PG)*). Chaque génération dépend de la durée de vie de l'objet créé. Bien évidemment, le JVM réserve les ressources de mémoire nécessaires pour un objet lorsqu'il rencontre la partie du code concernée. La taille assignée à ces générations influe énormément sur la performance du système. La figure 5.3 illustre l'effet de la section YG sur le délai.

Le test a été effectué trois fois avec différentes valeurs de YG. Un débit de 10 appels/s est généré pendant 50 secondes. Une rafale de 300 est arrivée juste après, et a duré 100 secondes. Le débit a diminué pour atteindre sa valeur initiale à la 150e seconde. L'algorithme de GC utilisé est le *Concurrent+parallèle*, la valeur maximale de la mémoire du JVM est fixé à 2G (*Heap*=2G). Dans chaque test la valeur maximale du YG est fixée à : 32 Meg, 512 Meg et 1G. Pour chacun des cas, les délais obtenus varient d'une façon différente. La figure 5.3 représente une comparaison entre les résultats obtenus.

YG=32 : Les délais ont suivi une variation plus faible. La moyenne durant la rafale était de 2757.27ms. La valeur maximale (3495.62ms) a été atteinte à la 65eme seconde. Ceci est causé par les pauses courtes et fréquentes du GC.

YG=512 : les pauses, dans ce cas, sont moins courtes et moins fréquentes par rapport au cas précédent. Les délais ont varié selon une moyenne de 2448.17 ms, la valeur maximale (4999.82 ms) a été atteinte à la 132eme seconde. Lorsqu'une pause est effectuée, le délai augmente d'une façon assez remarquable, cette augmentation est observable par l'usager. De plus, il y a des appels qui ont des délais acceptables (1717.5 à la 115éme seconde) tandis que

certains ont des délais inacceptables (4999.82 ms). À la fin de chaque pause les délais diminuent jusqu'à ce que le GC effectue une autre.

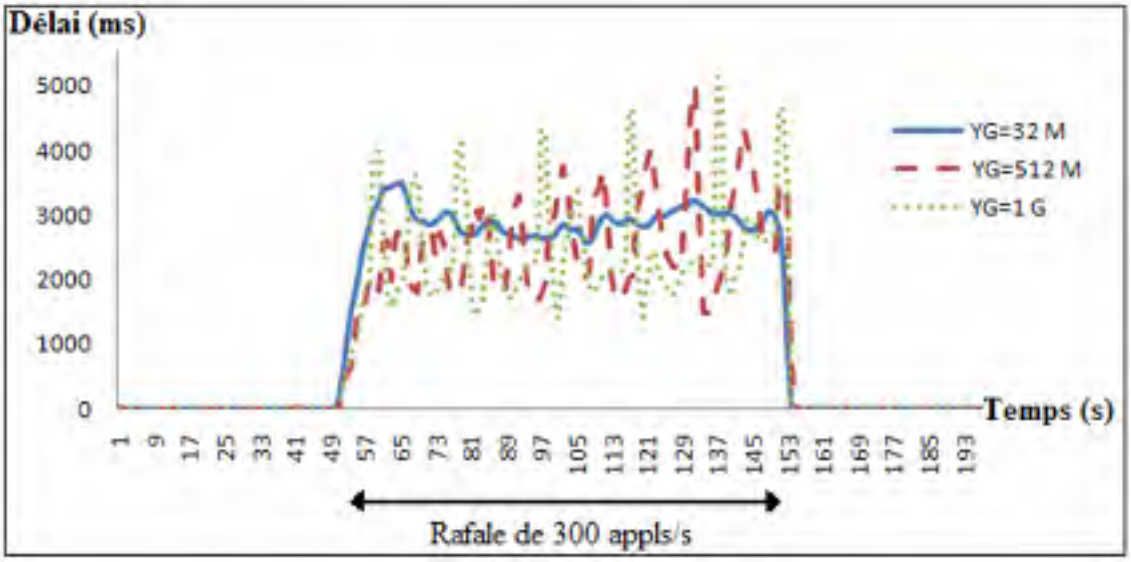

Figure 5.3 L'effet de l'YG sur les délais

YG=1G : lorsque YG est supérieur à *Heap*/2 , le système perd sa performance (Van Den Bossche et al., 2006). Ceci est remarqué dans la courbe lorsque YG=1G, où le délai a atteint des valeurs plus élevées (valeur maximale de 5146.34ms) et a augmenté d'une façon continue. Les délais ont varié selon une moyenne de 2300.34 ms. Les pauses étaient moins fréquentes et beaucoup plus longues.

Le délai, à la fin de la rafale, était supposé être plus grand que 4660.11 ms. Sauf que cette augmentation a coïncidé avec la chute du débit d'appel. Cette chute s'effectue d'une façon remarquable lorsque le GC n'est pas en mode d'exécution.

Lorsqu'une pause est effectuée, les délais subissent une augmentation importante, car la durée de la pause est beaucoup plus grande. La courbe montre que cette augmentation est graduelle, ainsi que la moyenne du délai pendant la rafale qui est inférieure à celle des cas précédents.

Cette différence n'est pas si importante (une centaine de millisecondes). De plus, les délais dépasseront les limites à un moment donné, notamment lorsque la période de rafale est beaucoup plus longue. Les délais, dans ce cas, deviennent inacceptables. Ce qui mène à une dégradation de QoS.

Il existe d'autres paramètres qui influent énormément sur les délais, sauf que ceci ne fait pas partie de l'objectif de ce travail. Pour minimiser les délais, il est recommandé de combiner les deux algorithmes *concurrent* et *parallèle*. De plus, précisez la valeur des YG et du *Heap* aboutit à une meilleure performance, car le temps de réallocation de mémoire sera minimisé (Van Den Bossche et al., 2006). Les paramètres utilisés dans les prochains tests étaient inspirés du (Van Den Bossche et al., 2006), où une configuration plus adaptée au délai a été proposée.

5.4 Le système FIFO-GFSIP

Le système FIFO-GFSIP traite les messages selon un mode : premier arrivé premier servi, ce qui veut dire que celui qui arrive en premier a la priorité d'être transmis au serveur SIP. Essentiellement, la priorité des messages est établie, non pas sur le type du message, mais selon le temps d'arrivé.

Étant donné que la longueur des files d'attente est finie, les messages qui arrivent quand les buffers sont pleins sont rejetés. D'autres sont ignorés. Ces deux opérations dans un système SIP sont tout à fait différentes. Un rejet signifie que l'émetteur devrait être notifié du rejet pour qu'il arrête la retransmission, tandis que le fait d'ignorer un message signifie que le UA continue à retransmettre jusqu'à la réception d'une réponse ou l'expiration d'un T_{max} (le T_{max} est défini dans le RFC 3261, il dépend d'un message). Cette opération est essentiellement contrôlée par UDP le protocole de transport utilisé.

Durant une rafale de trafic, la longueur des files d'attente joue un rôle très important sur le délai et le taux de rejet. Plus la longueur est grande, plus le délai augmente et le taux de rejet

diminue. La figure 5.4 montre l'effet de la longueur maximale de la file d'attente sur le délai :

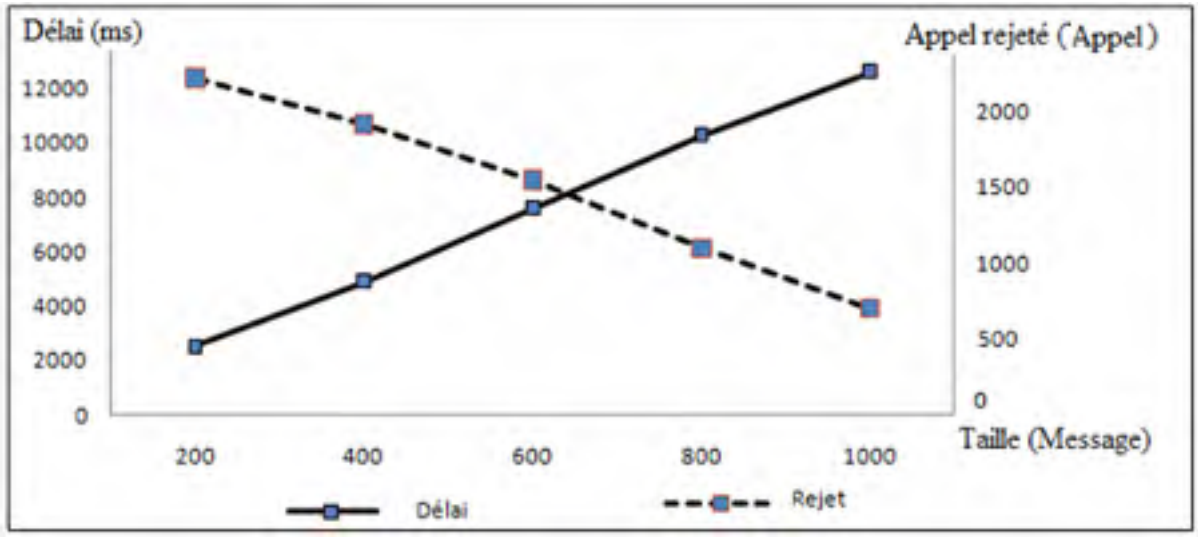

Figure 5.4 L'effet de la longueur des files d'attente sur le délai et le rejet

Le test a été initié par un débit léger de 10 appels/s qui a duré 50 secondes. Juste après la 50e seconde, le débit a augmenté de 200 appels/s en moins d'une seconde, la rafale a duré 100 secondes. À la fin de la 100e seconde, le débit est réduit à sa valeur initiale. La durée de sonnerie du téléphone est de 500ms et la conversation dans chaque appel dure 3 secondes. Le test a été effectué plusieurs fois afin de récupérer les résultats pour chaque longueur. Les résultats obtenus ne concernent que la période de rafale.

La figure 5.4 représente le délai moyen 'Invite-180' et '200-Ack' lorsque la file d'attente de 'Invite' atteint son maximum. Il est constaté que, plus la longueur augmente plus les délais deviennent importants, cela montre que les messages passent plus des temps dans les files d'attente. Au contraire pour le taux de rejet, il a subi une diminution, car la grandeur des files d'attente permet au taux d'admission d'appels d'être plus élevé.

La durée nécessaire pour que la file d'attente atteigne son maximum dépend de sa longueur. Avec une longueur de 200 messages, le maximum a été atteint à la 68eme seconde, tandis

que pour 1000 messages, le maximum a été atteint à la 125eme seconde, ce qui explique les résultats de rejet obtenus.

Il est aussi remarqué que les deux délais ont presque la même valeur. Ceci est dû à la discipline appliquée. Plus de détails suivront dans les prochaines sections.

5.4.1 L'analyse de la langueur des files d'attente

Afin d'analyser l'évolution des délais durant le test, la longueur maximale de la file d'attente de 'Invite' a été fixée à 200 messages. Dans ce cas (d'après la figure 5.4) les délais moyens durant la rafale doivent varier selon une moyenne de 2522.5ms. La file de 'Invite' a été choisie parce qu'elle influe sur la longueur de toutes les autres. En d'autres termes : si le GFSIP accepte un maximum de 200 appels, les autres files d'attente seront remplies par les messages subséquents des 200 messages 'Invite' qui ont été admis. La figure 5.5 montre les résultats du test.

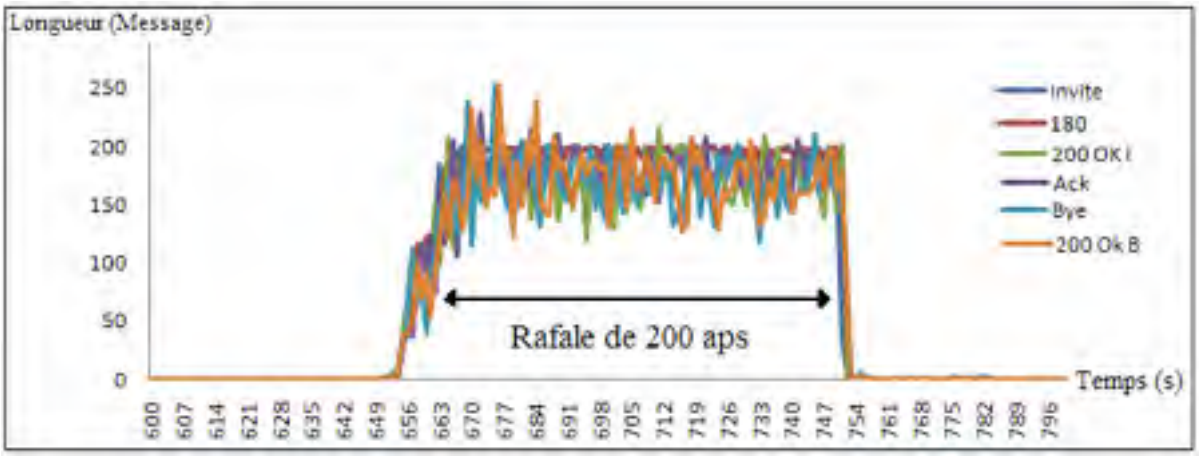

Figure 5.5 L'évolution de la longueur des files d'attente

L'augmentation de la taille des files d'attente apparaît clairement durant la rafale. Pendant les premières 500 millisecondes après le début de la rafale, seules les files d'attente de 'Invite' et celle des messages '180 Ring' commencent à se remplir. Juste après (500ms est la durée de la sonnerie) la taille des files d'attente '200 OK' et 'Ack' commence à augmenter

jusqu'à une valeur maximale de 209 messages, et finalement la rafale a atteint les files d'attente 'Bye' et '200 OK' après 3s (durée de la conversation) avec une valeur maximale de 253 message à la 675e seconde.

La discipline appliquée sur ces files d'attente est la cause principale sur l'évolution de leurs tailles. La rafale du message 'Invite' reçue ne fait qu'augmenter le nombre des messages subséquents, et donc, les messages qui arrivent vont toujours attendre ceux qui sont arrivés en premier. En conséquence, le nombre de messages dans les files devient de plus en plus important. La rafale a permis à la file d'attente de 'Invite' de se remplir rapidement. Le rejet a commencé juste après. Pour les autres files, la longueur varie d'une façon remarquable. Cette variation est due essentiellement à la discipline appliquée. La diminution de la longueur à un instant donné signifie que le GFSIP a traité un nombre important de message de la file d'attente en question. Les messages traités sont certainement arrivés en premier par rapport aux autres.

Il est remarqué (Figure 5.5) que parfois les tailles des files d'attente des messages subséquents dépassent 200 messages. Ce débordement est considéré négligeable et n'affecte pas trop le délai parce qu'il est limité par le contrôle d'admission. La cause de ce dépassement est la suivante : d'une part, le GIFSIP ne rejette pas les messages subséquents afin d'éviter les rejets inacceptables. Et d'autre part il y des moments où la taille d'une file grandit continuellement parce que l'ordonnanceur arrête le traitement de ses messages et priorise d'autres qui sont arrivés en premier.

Il y a aussi des moments où la taille des files des messages subséquentes reste au-dessous de 200 messages. Ceci est expliqué par le phénomène de dépendance entre messages vu précédemment. Il y a des instants (millisecondes) où l'ordonnanceur traite quelques dizaines de messages de la même file d'attente, le moment où il arrête de traiter les messages dont cette file dépend explicitement.

Lorsque toutes les files d'attente se remplissent, l'admission des messages 'Invite' contrôle explicitement la taille de la file des messages 'Invite' tandis qu'elle limite implicitement la taille des autres files. Ceci est illustré dans la figure 5.5 où la longueur des files varie au-dessous de 200 messages dans la plupart du temps.

5.4.2 L'analyse des délais et rejets

Il est clair que l'augmentation de la taille des files durant la rafale augmentera les délais qui en dépendent. Les figures 5.6 et 5.7 représentent l'évolution des délais 'Invite-180' et '200-Ack' durant toute la période du test :

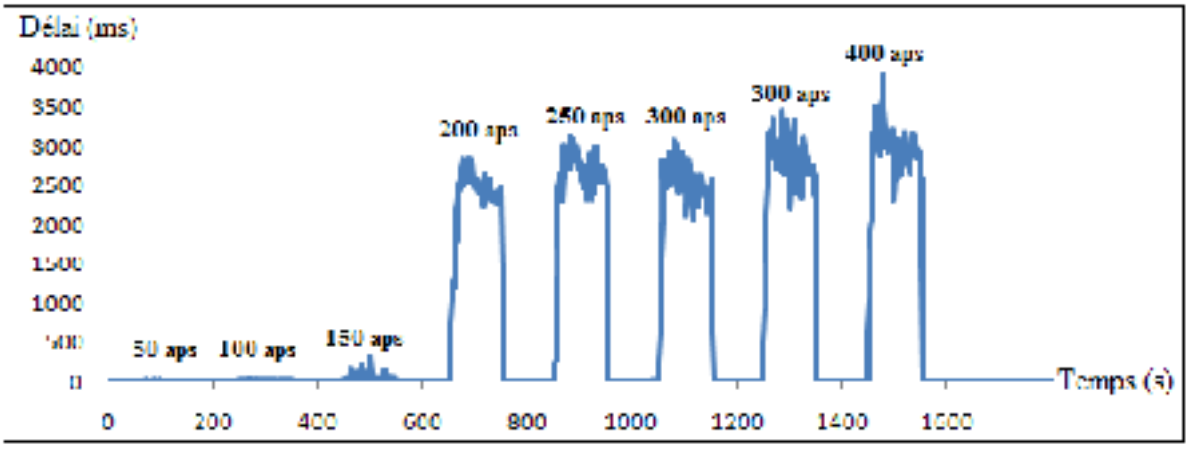

Figure 5.6 Le délai 'Invite-180'

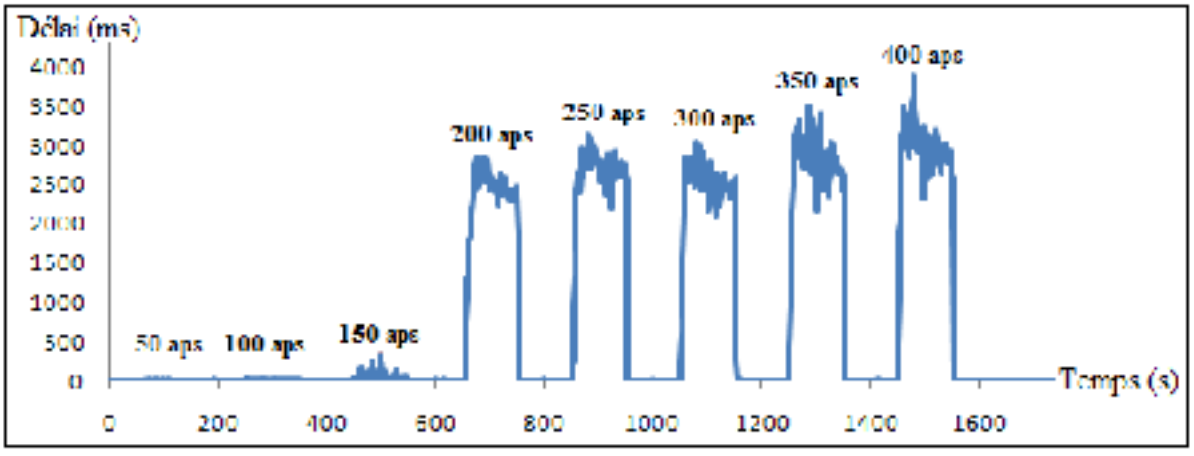

Figure 5.7 Le délai '200-Ack'

Lorsque le débit d'appels est faible, les deux types de délai varient selon une valeur négligeable. Le GFSIP a toutes les ressources nécessaires pour supporter ce débit sans générer de délai. Il est constaté que même à un débit de 150 appels/s, les délais sont restés très faibles avec une valeur maximale de 314 ms qui a été atteinte une seule fois à la 501e seconde.

Lorsque les rafales ont dépassé 200 appels/s, le débit d'entrée a dépassé la vitesse de traitement du GFSIP. Ceci a causé une augmentation dans le nombre de messages en attente, d'où une augmentation des délais.

Durant les rafales, les deux délais varient selon une moyenne dépendante du débit d'appel. Il a été constaté que l'augmentation du débit de la rafale implique une petite augmentation dans la moyenne des délais. Le délai moyen durant la rafale de 200 appels/s est de 2233.4 ms tandis que pour la rafale de 400 appels/s la moyenne est de 2856.6 ms. Cette augmentation est due essentiellement à l'utilisation additionnelle des ressources par les appels admis et rejetés. Ceci confirme que le coût de rejet d'appel n'est pas négligeable.

La figure 5.8 illustre une représentation comparative entre le délai 'Invite-180' et '200-Ack'. Les deux courbes paraissent superposées parce qu'il y une différence de 500 ms (durée de la sonnerie) entre le début de la rafale de 'Invite' et le commencement de la rafale du message '200 Ok (Invite)'. Cette différence n'est pas visible dans la courbe. De plus, les deux délais varient selon la même moyenne.

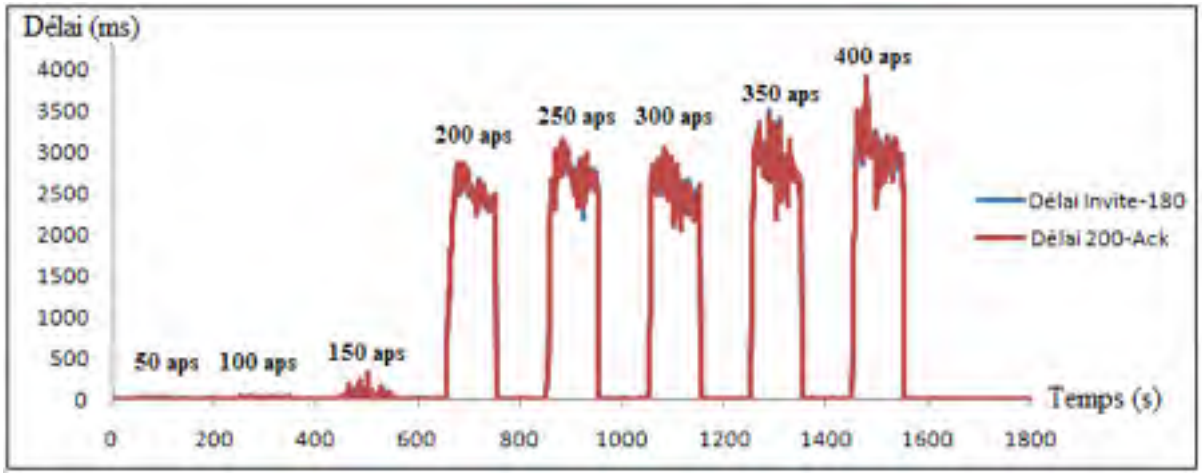

Figure 5.8 Délai 'Invite-180' vs délai '200-Ack'

Durant la rafale, quel que soit le message qui arrive, il trouvera un nombre de messages bien précis déjà en attente. Selon la discipline FIFO, ce message doit attendre le traitement de tous les autres messages avant d'arriver à l'ordonnanceur. Tous les messages qui arrivent subiront le même temps d'attente, ce qui explique la valeur égale des deux types de délais, d'où la superposition des courbes illustrée dans la figure 5.8.

L'envoi du message '100 Trying',\ a permis d'éviter la retransmission d'un nombre important du message 'Invite' durant la rafale. Outre, le mécanisme de détection de retransmission a joué un rôle très important dans la diminution des délais. Ce mécanisme a été introduit dans d'autres travaux de recherche et a démontré une efficacité dans la diminution des délais d'attente. Dans ce projet, il a été implémenté pour le combiner avec l'approche proposée. En résultat, il a permis aux files d'attente d'héberger que les messages non dupliqués ce qui a permis la diminution au niveau de temps d'attente. Pour plus d'information sur l'effet de la retransmission sur le délai, il est recommandé d'avoir recours aux résultats obtenus dans (Jing et al., 2007).

À la fin de chaque rafale, le débit chute à 10 Appels/s, l'ordonnanceur a eu le temps de vider toutes les files d'attente ce qui a diminué les délais. La chute rapide de délai est due au mécanisme de détection de retransmission, les messages n'ont pas à attente à ce que les

messages dupliqués arrivent à l'ordonnanceur. Ils se sont traités tout de suite. Ce qui confirme les résultats obtenus dans (Jing et al., 2007).

La figure 5.9 présente le nombre d'appels rejetés durant le test. Le rejet d'appel dépend intimement du débit. Tant que la longueur de la file d'attente de 'Invite' est au-dessous de 200 messages, les appels sont admis, au fur et à mesure que le nombre des messages subséquents augmente, la fréquence de traitement des messages 'Invite' diminue. Lorsque la file d'attente de 'Invite' atteint son maximum, le GFSIP commence à envoyer les messages '503 Services Unavailable' en fonction du nombre d'appels qui sont déjà admis.

Le rejet a commencé lorsque la rafale a atteint 200 Appels/s avec 2212 appels rejetés (11.06 %). Au fur est à mesure que le débit augmente, le pourcentage des appels rejeté augmente d'une façon dramatique. Lorsque la rafale a atteint 400appels/s, 21150 appels ont été rejetés avec un pourcentage de 52.87 %. C'est qui est un taux très élevé.

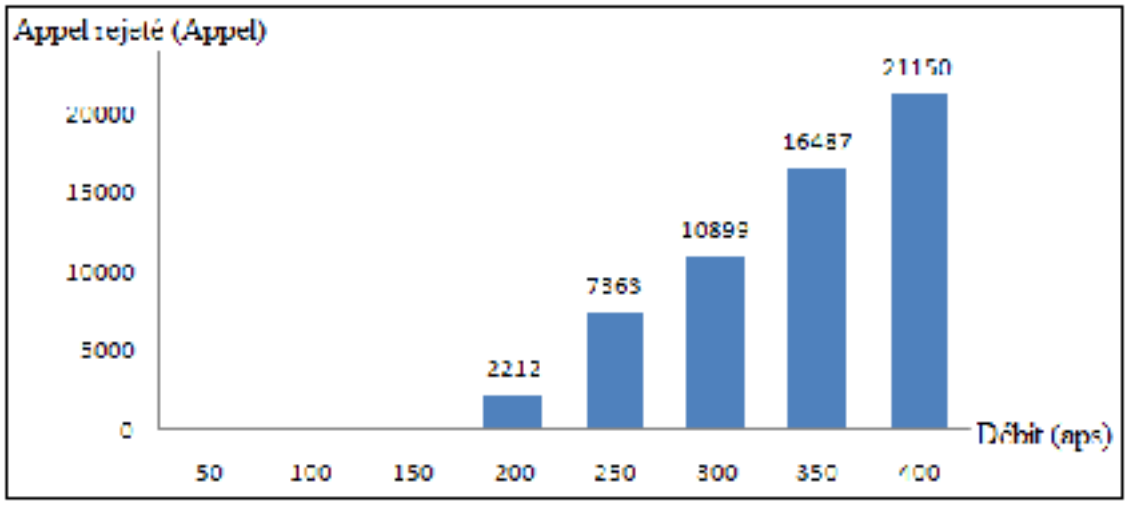

Figure 5.9 Le nombre d'appels rejetés

Le GFSIP devrait avoir plus de ressources afin de supporter les débits élevés. Le nombre d'appels reçus lorsque la rafale a atteint 400Appel/s a généré un nombre très grand de messages subséquents. Les files d'attente de ces messages étaient dans la plupart du temps plein, ce qui a influé sur le taux d'admission des messages 'Invite'. De plus, l'augmentation

des délais d'attente discutée auparavant avait un impact additionnel sur l'admission des appels.

Avec un taux de rejet de 52.87 % (en le comparant à 11.06 %), il est évident que le coût de rejet des appels a réduit la performance du GFSIP. De plus, vu les délais obtenus, le nombre de messages retransmis a augmenté dramatiquement lorsque le débit de la rafale a atteint 400Appel/s. La détection de ces messages avait un coût additionnel qui a influencé sur le taux d'admission.

Le rejet d'appel est interrompu dès le début de la chute du débit d'appel. Le débit d'entrée a diminué par rapport au débit de traitement. Ainsi le nombre de messages 'Invite' a diminué, ce qui a permis à tous les appels reçus d'être admis.

Pour conclure, le GFSIP a reçu 170000 appels pendant les neuf rafales, dans lesquelles 58111 appels ont été rejetés (34.18 %). Ce résultat nécessite une comparaison avec les autres disciplines. Une analyse comparative sera effectuée dans l'une des prochaines sections.

5.4.3 L'effet de la longueur maximale sur la tolérance

Le choix de la longueur maximale de la file d'attente 'Invite' est la méthode utilisée pour profiter du délai peu acceptable discuté dans le chapitre précédent. Il a été montré que le délai 'Invite-180' est un délai peu acceptable, qui se compose de deux délais de tolérances différentes. Il a été aussi démontré que le délai du message 'Invite' est le délai adéquat dont il est possible de profiter afin d'augmenter le nombre d'appels admis.

Le choix du délai dépend de l'administrateur et du type de la clientèle servie. Ceci est un choix entre la tolérance aux délais et celle aux rejets. Si le délai 5s est considéré acceptable, dans ce cas le GFSIP sera configuré avec une file d'attente 'Invite' de 400 messages. Ce qui permettra au taux de rejet de diminuer par rapport à une file d'une taille maximale de 200 messages. Par contre, si la valeur de 5s est considérée acceptable pour un délai 'Invite-180',

il est clair qu'elle ne l'est pas pour le délai '200-Ack'. Ainsi, la discipline appliquée est apparue inconvenable pour les files d'attente longues. Ce qui a suscité le besoin de tester d'autres disciplines.

5.5 Le système FQ-GFSIP

L'analyse de l'effet des disciplines sur les délais est parmi les objectifs essentiels de ce projet. Cette partie introduit l'application d'une autre discipline intitulée FQ (Fair Queuing) qui se base sur le traitement équitable de toutes les files d'attente sans aucune priorité. Dans le cas du FIFO, les messages qui arrivent en premier étaient priorisés, il y avait des moments où l'ordonnanceur traite un nombre de message dans la même file d'attente sans passer aux autres. Lorsqu'il finit, il passe à une autre puis il traite un minimum de 1 message. Il y avait des moments où quelques files d'attente ne sont pas consultées par l'ordonnanceur depuis une période temps. Ce qui a engendré un traitement inéquitable des files d'attente en question.

La discipline FQ a pour but d'éviter le traitement inéquitable des files d'attente. En appliquant cette discipline, l'ordonnanceur retire les messages d'une manière équitable sans aucune priorité (Round-Robin). Il effectue un traitement selon un ordre préétabli avec un maximum d'un seul message par itération.

Dans chaque itération, l'ordonnanceur consulte la file d'attente une seule fois et retire un message pour l'envoyer au serveur SIP. Le test effectué est semblable à celui appliqué pour FIFO-GFSIP. Les résultats obtenus concernent les délais, les rejets et la longueur des files d'attente.

5.5.1 L'analyse des résultats

Les résultats obtenus avec un FQ sont différents par rapport au FIFO. Une grande différence entre les délais a été constatée durant les rafales. Les figures 5.10 et 5.11 présentent les résultats des délais obtenus.

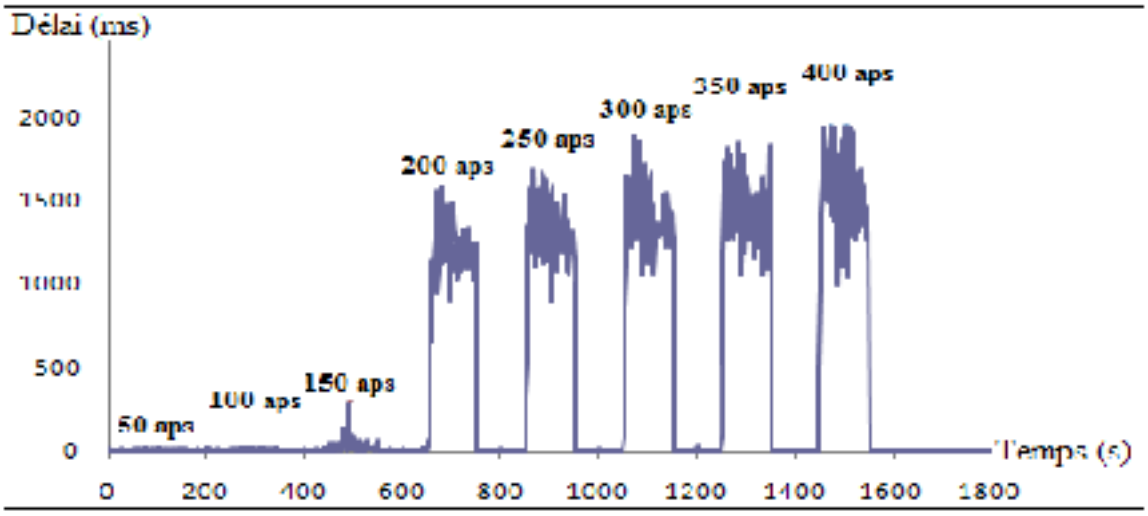

Figure 5.10 Le délai 'Invite-180'

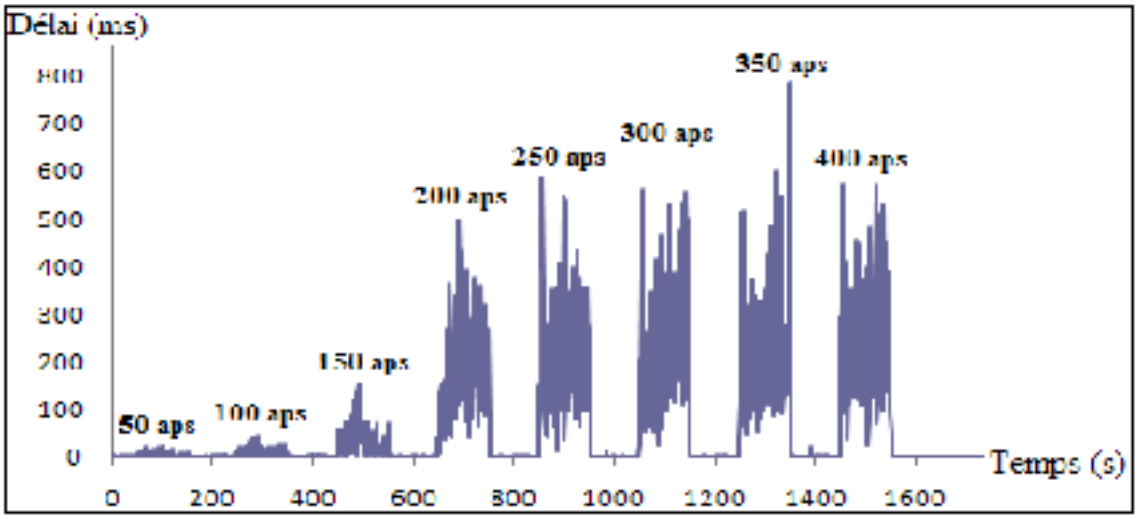

Figure 5.11 Le délai '200-Ack'

Toujours dans les premières 600 secondes, les délais étaient assez faibles. Le GFSIP était capable de supporter jusqu'à 150 Appels/s sans générer des délais importants. Le délai 'Invite-180' a atteint une valeur maximale de 114.817 ms à la 485e seconde une seule fois.

Durant la rafale de 200 Appels/s, les délais ont varié selon une moyenne de 1104.4936 ms. Cette moyenne a augmenté en fonction du débit. Durant la rafale 400 Appel/s, la moyenne a atteint 1539.8191ms.

En ce qui concerne le délai '200-Ack', il a atteint une valeur maximale de 154.671 ms à la 493e seconde. Les délais ont varié selon une moyenne de 222.87 ms durant les rafales au-delà de 150 Appels/s.

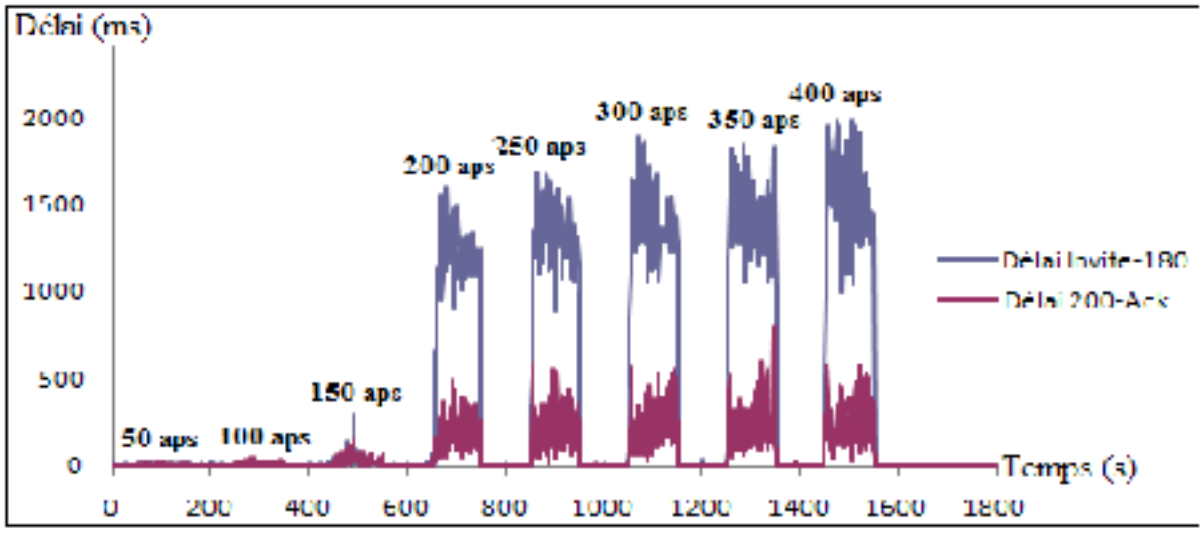

Figure 5.12 Le délai 'Invite-180' vs délai '200-Ack'

La figure 5.12 présente une comparaison entre les délais 'Invite-180' et '200-Ack'. La différence entre les deux délais est essentiellement due à la discipline appliquée. La manière dont l'ordonnanceur retire les messages pour les envoyer au serveur SIP a permis au délai '200-Ack' d'être beaucoup plus bas par rapport au délai '180-Invite'.

Au début de chaque rafale, le GFSIP continue à accepter les appels jusqu'à ce que la longueur de la file d'attente de 'Invite' atteigne sa valeur maximale (200 messages). Au même

temps, il transmet au serveur SIP les messages subséquents des messages Invite' qui ont été admis.

Le phénomène de dépendance entre messages SIP explique facilement les résultats obtenus. L'admission des messages "Invite" influe énormément sur le débit des messages subséquents. L'ordonnanceur consulte les en-têtes de chaque file d'attente d'une façon équitable, d'où la probabilité de traitement des six messages reste égale durant toute la période de test. En résultat, pour chaque message 'Invite' traité, ses messages subséquents sont traités tout de suite, ce qui explique les résultats obtenus.

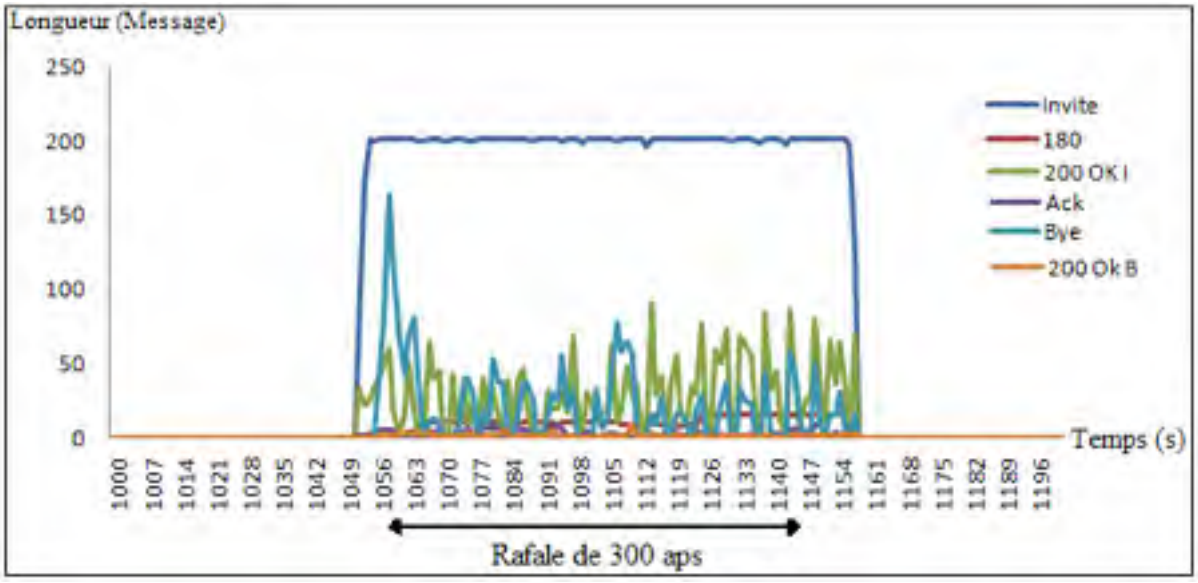

Figure 5.13 L'évolution de la taille des files d'attente

La figure ci-dessus présente la variation de la longueur des files d'attente durant une rafale de 300 Appels/s. En dehors de la rafale, les files d'attente étaient pratiquement vides. Durant la rafale, les files d'attente des messages '180 Ring' , 'Ack' ont atteint des valeurs maximales très faibles (respectivement : 16 messages, 9 messages) par rapport aux autres messages. Ce qui est attendu des messages qui ont des dépendances explicites complètes. La file du message 'Invite' a atteint son maximum (200 messages), la file de '200 Ok (Invite)' a varié entre 84 et 1 message. La file d'attente de 'Bye' a atteint une valeur maximale de 164

messages à cause d'un grand nombre de messages qui sont arrivés tous à la fois juste après la durée de la conversation (3s).

La variation de la longueur des files d'attente 'Bye' et '200 Ok (Invite)' est due essentiellement à la dépendance implicite aux messages qui leur précédent. Les moments où la longueur des files diminue signifient que le débit de traitement était plus élevé par rapport au débit d'entrée. Ce dernier dépend intimement du traitement des messages précédents. Le débit de traitement du message '200 OK (Invite)' dépend du taux d'admission des messages 'Invite'. C'est pour cette raison que la courbe a subi des variations durant toute la période de rafale. La même raison explique la variation de la file d'attente du message 'Bye'.

Un grand pourcentage du délai 'Invite-180' obtenu, est dû essentiellement au délai d'attente des messages 'Invite'. La file d'attente du message '180 Ring' avait une longueur très faible pendant toute la période de rafale. Lorsqu'un message 'Invite' est admis, son délai ne dépendra pas seulement des messages qui étaient déjà dans les files (contrairement au FIFO-GFSIP). Mais au fur et à mesure que les messages 'Invite' sont transmis leurs messages subséquents le sont aussi. Ainsi, un nouveau message 'Invite' doit attendre le traitement d'un maximum de 6 x 200 messages = 1200 messages avant d'être servi.

En comparant ce délai à celui des messages '180 Ring', ce dernier est considéré négligeable. Les messages '180 Ring' sont traités rapidement lorsqu'ils sont mis dans la file d'attente. La dépendance explicite complète entre ces deux messages permet aux messages '180 Ring' d'arriver selon le débit de traitement des messages 'Invite', et donc, lorsqu'un message 'Invite' est traité, sa réponse '180 Ring' est traitée tout de suite, car l'ordonnanceur consulte les files d'attente d'une façon périodiques et équitable.

L'allure de la courbe du délai '200-ack' est justifiée par trois facteurs essentiels remarqués lorsqu'un FQ-GFSIP est utilisé. D'une part la durée de sonnerie qui est égale pour tous les appels durant toute la période du test, d'autre part la dépendance explicite partielle entre les

messages ‘Invite’ et ‘200 OK’. Troisièmement, la dépendance explicite complète entre les messages ‘200OK’ et ‘Ack’.

Une grande partie du délai ‘200-ack’ dépend du délai d’attente du message ‘200 OK’ dans la file d’attente du GFSIP. Ceci est expliqué par la dépendance explicite complète entre les deux messages. Plus le débit de traitement des messages ‘200 OK’ augmente, plus le débit de réception des messages ‘Ack’ augmente aussi. Ces derniers sont traités tout de suite grâce à la discipline utilisée.

Il a été constaté que les messages ‘200 OK’ passent plus de temps dans la file d’attente dépendamment du temps de la sonnerie (TS). Ce message arrive en rafale parce qu’il est considéré que tous les utilisateurs décrochent ‘pratiquement’ tous à la fois. Ainsi, plus le temps de sonnerie est grand, plus le volume de la rafale ‘200 Ok (Invite)’ devient important. Afin d’investiguer davantage sur l’effet du TS sur le délai ‘200-Ack’, un test avec différentes valeurs de TS a été effectué.

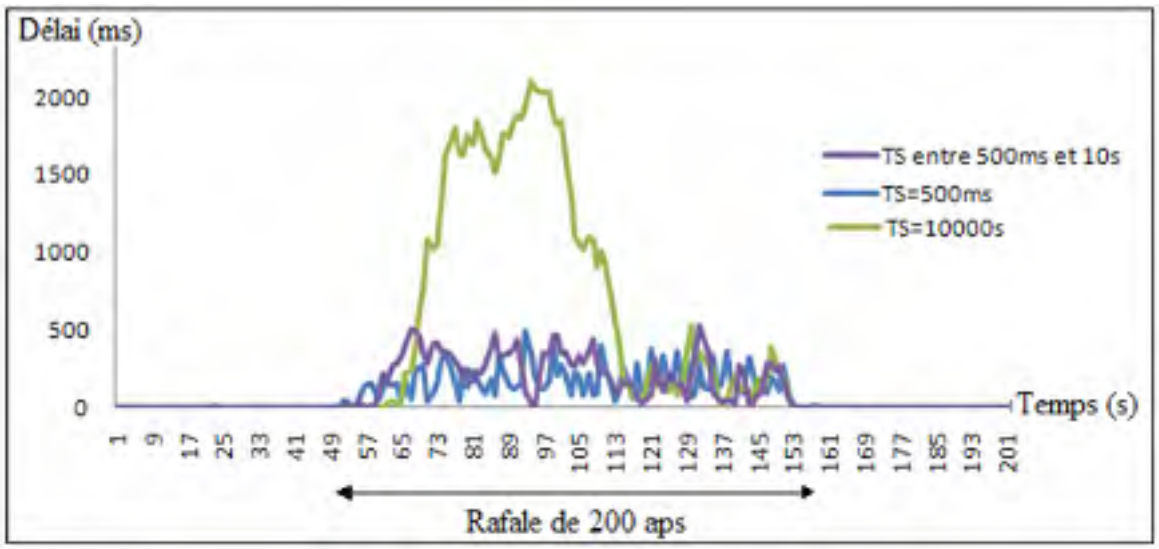

Figure 5.14 L’effet du TS sur le délai ‘200-Ack’

La figure 5.14 présente les délais ‘200-Ack’ obtenus à l'utilisation de trois distributions de TS différentes. Le TS=500ms est une distribution fixe dans laquelle tous les appels générés ont

la même valeur TS. Cette distribution est celle utilisée dans tous les tests effectués. Le délai obtenu est assez faible (moyenne = 1092.18 ms). Les messages '200 OK' ne restent pas très longtemps dans la file d'attente. Le GFSIP était capable de traiter les messages qui sont arrivés après 500 ms sans délai apercevable.

Le TS=10000ms est une distribution fixe de temps de la sonnerie où les usagers décrochent après 10 s. Le délai maximal obtenu pendant la rafale est 2100 ms. Pendant les 10 premières secondes, toutes les files d'attente étaient vides à part celle du message ‘180 Ring’ et ‘Invite’ (Dépendance explicite complète). Juste après, une grande rafale de message ‘200 OK’ est arrivée presque tous à la fois. En conséquence, la file a été remplie ce qui a engendré une augmentation de délai. Au fur et à mesure que les ‘200 OK’ sont traités, d’autres messages subséquents arrivent. Cela a diminué le taux d’admission des messages ‘Invite’. Ceci a permis au GFSIP de vider la file d’attente du message ‘200 OK’, d’où une chute importante du délai a été effectuée.

Afin de s'approcher plus de la réalité, la valeur TS a été distribuée aléatoirement durant la période du test. Sa valeur variait entre 500 ms et 10 000 ms. Le délai obtenu (moyenne de 1107.562 ms) était assez faible parce que la distribution utilisée contribue à un certain lissage du flux ‘200 OK’. cela l’a empêché d'arriver en rafale, d'où le fait que le GFSIP a eu la possibilité de les traiter plus rapidement.

Cette expérimentation a montré l'effet de temps de la sonnerie du téléphone lorsqu'un FQ-GFSIP est utilisé. Il est donc important de le mentionner lorsque ce type de test est effectué.

5.5.2 Les rejets

Le traitement rapide des messages subséquents a réduit le débit de traitement des messages ‘Invite’ pendant le début de la rafale. Ce qui a permis à sa file d’attente d’atteindre son maximum rapidement.

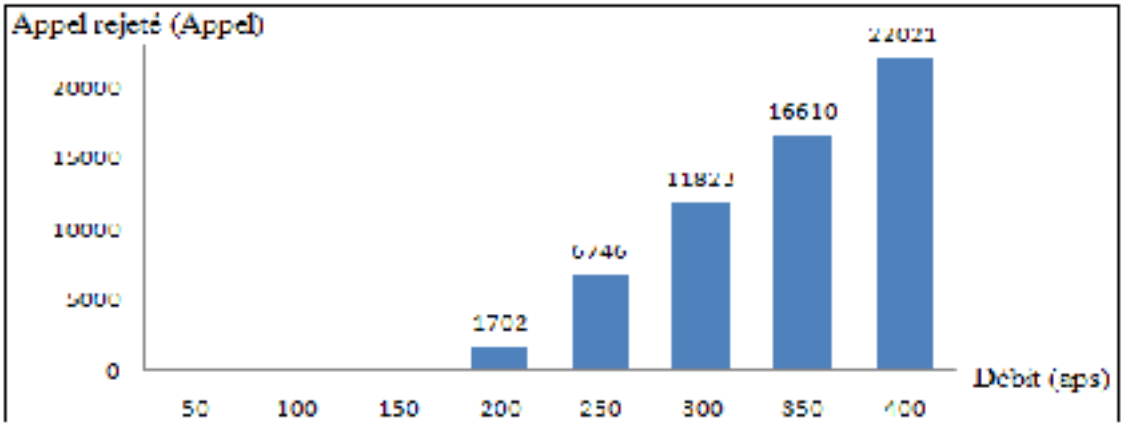

Figure 5.15 Le nombre d'appels rejeté

Le taux d'admission influe énormément sur le taux de rejet, car pour un seul message 'Invite' admis, cinq autres messages seront reçus (les messages subséquents). Et donc plus le taux d'admission augmente plus les nombres des messages subséquents augmentent. En conséquence, les périodes qui suivent auront un taux de rejet plus élevé.

Selon la figure 5.15, le rejet a commencé dès le début de la rafale de 200Apple/s. 1702 appels ont été rejetés avec un pourcentage de 8.51 %. Ce pourcentage a augmenté jusqu'à la rafale de 400Appels/s ou 22021 appels ont été rejetés (55.05 %). Comme c'est déjà expliquée, cette augmentation est, essentiellement, due à la consommation additionnelle des ressources.

Le GFSIP a reçu 170000 appels et 58902 appels ont été rejetés (34.64 %). Une comparaison entre les valeurs obtenues dans chacun des GFSIP sera effectuée dans la dernière section.

5.6 Le système PQ-GFSIP

Ce système implémente une certaine priorité entre les messages SIP. Quelques files d'attente seront plus priorisées que d'autres afin de respecter les tolérances discutées dans les chapitres précédents. Avant de commencer l'analyse des résultats, le principe de priorité établi sera discuté en premier.

5.6.1 La priorité des messages SIP

Une session SIP est sous forme d'échange de message entre UA menant à d'établir une session média, dépendamment du service offert. Afin de garantir des délais acceptables et minimiser le nombre de rejets, il fallait établir une certaine priorité entre ces messages.

La priorité des messages permettra au GFSIP de traiter les messages critiques en premier ce qui permettra d'éviter les délais inacceptables. Les messages avec basse priorité seront traités plus tard lorsque le GFSIP aura la possibilité.

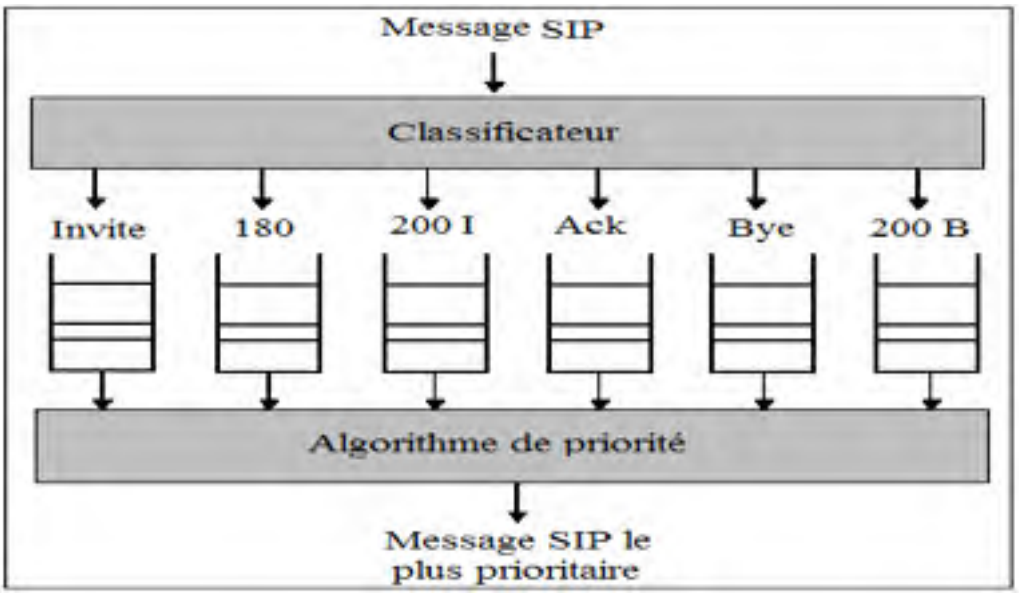

Figure 5.16 Le mécanisme de gestion des priorités

La figure 5.16 présente le mécanisme de la gestion prioritaire des messages. Lorsque le message est admis, il sera mis dans la file d'attente appropriée. Un algorithme de priorité est logiquement situé à l'en-tête des files d'attente. Cet algorithme est basé sur un Thread qui consulte périodiquement les en-têtes de chaque file d'attente pour récupérer le message le plus prioritaire. Le degré de priorité établi est illustré dans le tableau 5.1.

Tableau 5.1 Priorité des messages SIP

Message	Priorité	Tolérance au délai
Invite	3	Peu acceptable
180 Ring	2	Peu acceptable
200 OK (Invite)	1	Inacceptable
Ack	0	Inacceptable
Bye	4	Tolérable
200 OK (Bye)	5	Tolérable

La priorité établie se base sur l'engagement dans le processus d'établissement d'appel. En d'autres termes, plus on s'engage dans l'établissement de l'appel, plus les messages auront plus de priorité. La signalisation de terminaison d'appel sera moins prioritaire.

Le message 'Ack' est le message le plus prioritaire (0), Il en est ainsi parce qu'il est le dernier message échangé pour que la conversation téléphonique commence. Le délai de ce message est considéré inacceptable, et donc il sera servi dès sa réception.

Le message 200 Ok (Invite) est doté d'une priorité de deuxième degré, son délai est aussi considéré inacceptable, le GFSIP doit le traiter dès sa réception. Ce message est considéré critique parce qu'il peut arriver en rafale, ce qui engendra des délais additionnels pour les messages moins prioritaires.

Le choix de la priorité du message '180 Ring' peut dépendre de l'administrateur et de la clientèle servie. S'il est acceptable d'établir des communications sans que l'appelant entende la tonalité de sonnerie, ce message peut avoir la priorité (3), sinon, il aura une priorité de troisième degré (2). Cette implémentation a donné plus d'importance à ce message.

Le message 'Invite' a une priorité de 4e degré (3). Les appels déjà en cours sont plus prioritaires que les nouveaux appels. Les messages 'Bye' et '200 Ok (Bye)' ont

respectivement les priorités (4) et (5). Ces messages ont la priorité la plus basse puisqu'ils ne servent qu'à terminer l'appel.

La figure 5.17 montre l'algorithme implémenté pour garantir la priorité des messages expliqués ci-dessus. Son code se trouve dans l'annexe 5. Un thread de type 'Daemon' lit les en-têtes de chaque file d'attente. Il est censé vider la file d'attente prioritaire avant de passer à l'autre. L'exécution du thread est sous forme d'une boucle qui suit le même ordre les files d'attente dans chaque itération. La boucle commence toujours par la vérification du message le plus prioritaire 'Ack'. Si sa file d'attente est vide, le thread passe à la file d'attente '200 OK (Invite)'. Il retire le premier message de l'en-tête, l'envoi au serveur SIP et recommence la boucle. Le même principe est appliqué sur les messages à basse priorité.

Cette méthode permettra de vider les files d'attente les plus prioritaires avant de passer aux autres, elle permettra aussi de traiter les messages de haute priorité le plus rapidement possible. En résultat : le GFSIP aura pour objectif de garder les files d'attente prioritaires vides pendant toute la période de rafale. Tandis que pour la file d'attente de 'Invite', il fera du 'Best effort' vu le débit qu'il reçoit durant cette période.

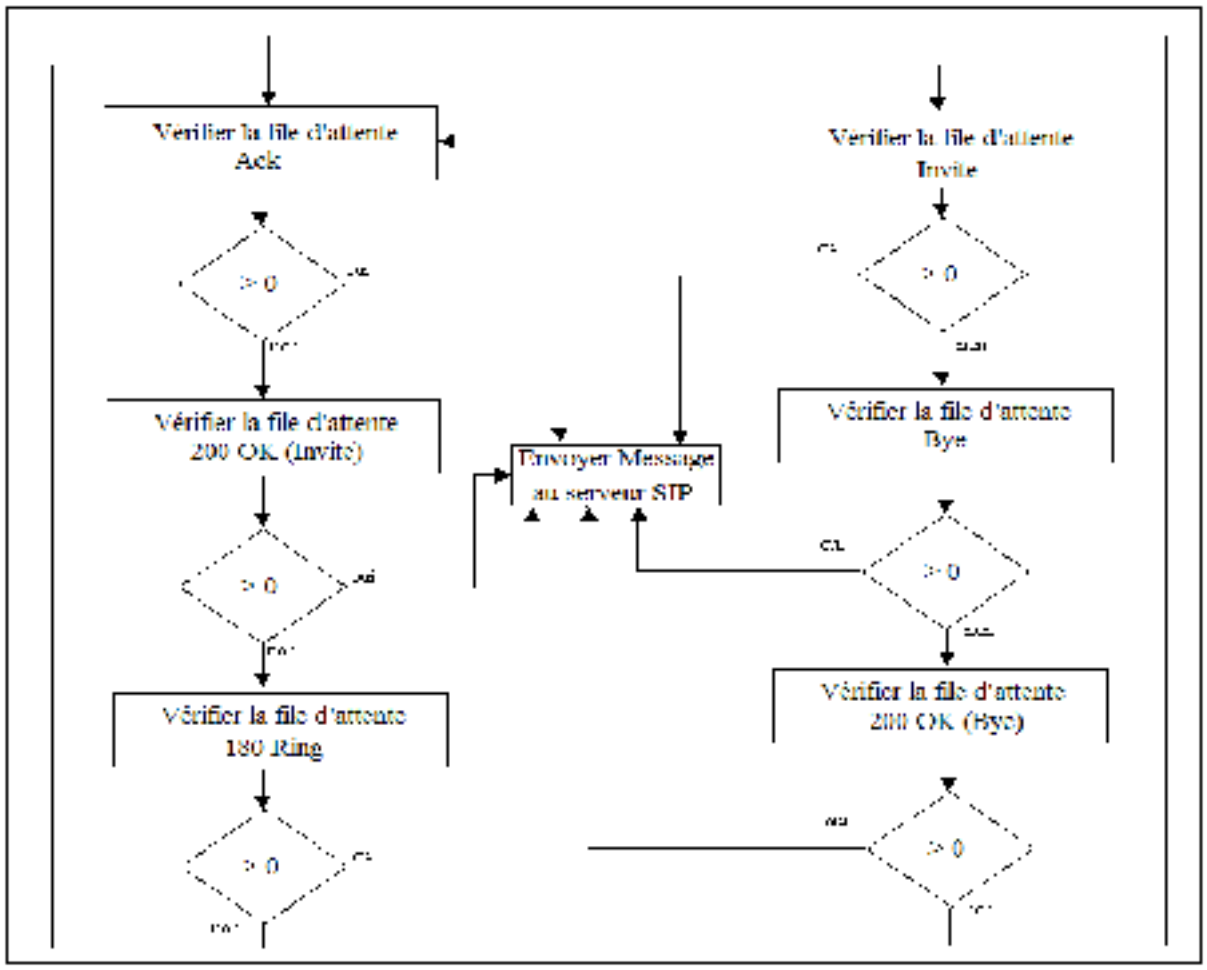

Figure 5.17 L'algorithme de priorité implémenté

Cette priorité a pour but de minimiser au maximum le délai '200-Ack', et profiter des délais acceptables pour minimiser les délais peu acceptables et augmenter le taux d'admission. En résultat, le nombre d'appels rejetés devrait être minimisé.

5.6.2 L'analyse des résultats

La priorité des messages établie a joué un rôle très important dans la diminution des délais '180-Invite' et '200-Ack'. Les figures 5.18 et 5.19 présentent les résultats obtenus.

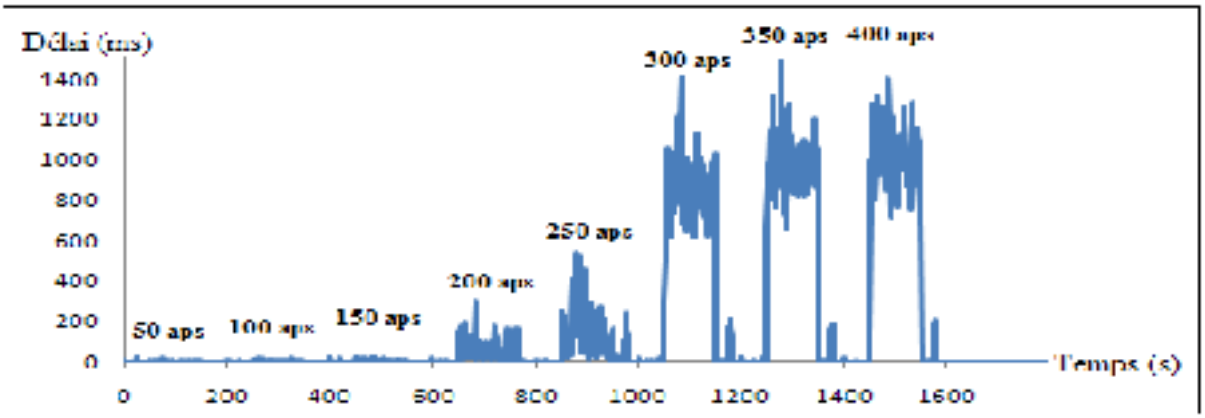

Figure 5.18 Le délai '180-Invite'

Une grande amélioration a été constatée concernant le délai 'Invite-180'. Sa valeur était beaucoup plus faible pour les deux rafales successives 200 Appels/s et 250 Appels/s. La moyenne était respectivement de 57.31ms et 153.42 ms, ce qui n'était pas le cas pour les deux autres systèmes.

Le fait de prioriser le message 'Invite' par rapport aux messages 'Bye' et '200 OK (Bye)' a augmenté la vitesse de son traitement. L'ordonnanceur retire le message avec une fréquence beaucoup plus grande. En résultat : les délais d'attente ont été minimisés.

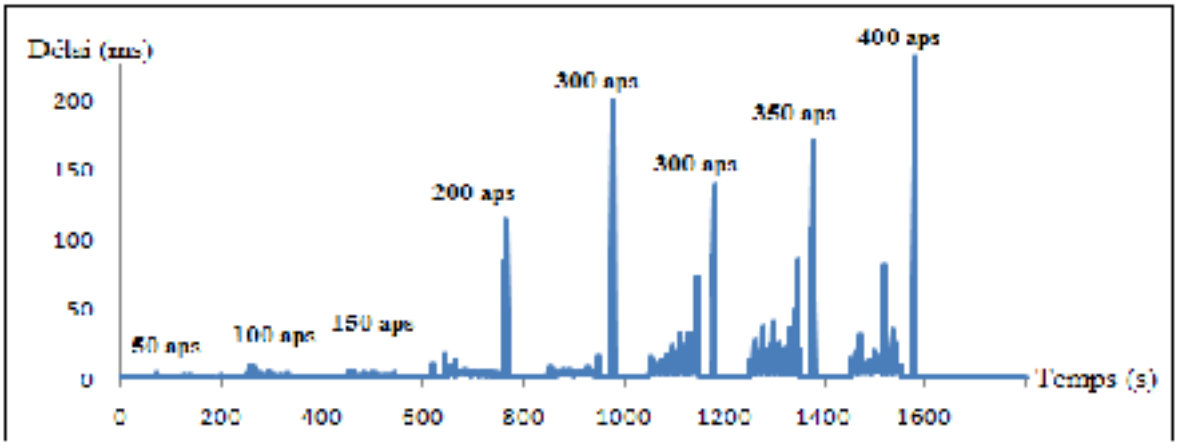

Figure 5.19 Le délai '200-Ack'

Une amélioration au niveau des délais a été constatée durant toute la période du test. Les délais ont été minimisés durant les neuf rafales déclenchées. Ce qui prouve la performance de la méthode implémentée.

Les délais '200-Ack' ont, également, subi une diminution remarquable. Une petite augmentation est constatée à partir de la rafale de 300 Appels/s. Mais elle reste trop faible, et n'est pas apercevable par l'usager. Le GFSIP a été capable de traiter les messages '200 OK' et 'Ack' le plus rapidement possible. Normalement, les délais sont supposés être plus bas que la valeur obtenue. Toutefois, la GC et le coût de rejet ont un grand effet à cet égard.

Les sauts de délai qui existent à la fin de chaque rafale sont causés principalement par les pauses du GC. Car la rafale n'est pas vraiment terminée. La file d'attente de 'Bye' et '200 OK' étaient toujours pleines et le GFSIP a commencé leur traitement juste après. Ce traitement avaient avait un effet relatif sur les messages prioritaires. Toutefois, il reste négligeable puisqu'il augmente de quelques centaines de millisecondes pour une période très courte.

La figure 5.20 présente une comparaison entre les deux délais en question. La différence apparaît d'une façon très claire. L'effet de la discipline appliqué montre clairement que les messages '200-OK' et 'Ack' ont été traités avec un minimum de retard. Ce qui est le but de cette implémentation.

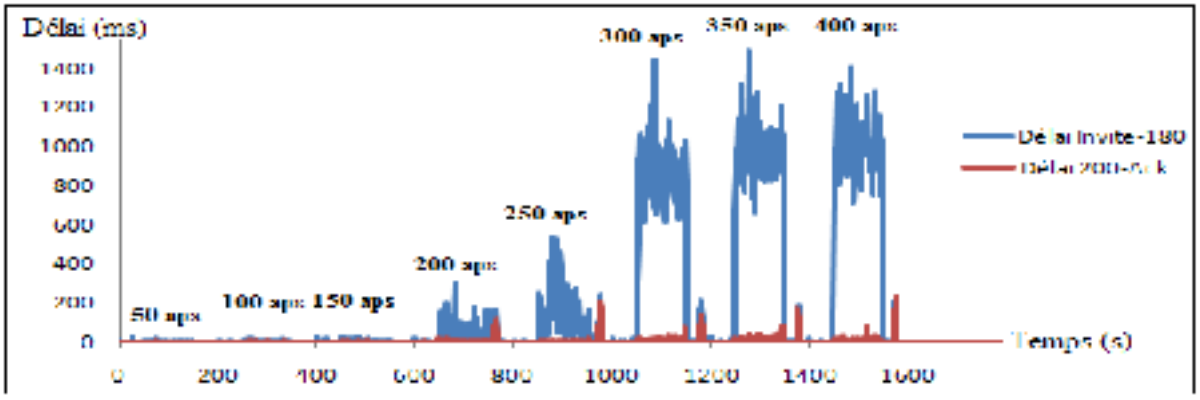

Figure 5.20 Le délai 'Invite-180' vs '200-OK'

La méthode de priorité implémentée a permis une grande amélioration concernant le nombre d'appels rejetés. La figure 5.21 présente les résultats obtenus.

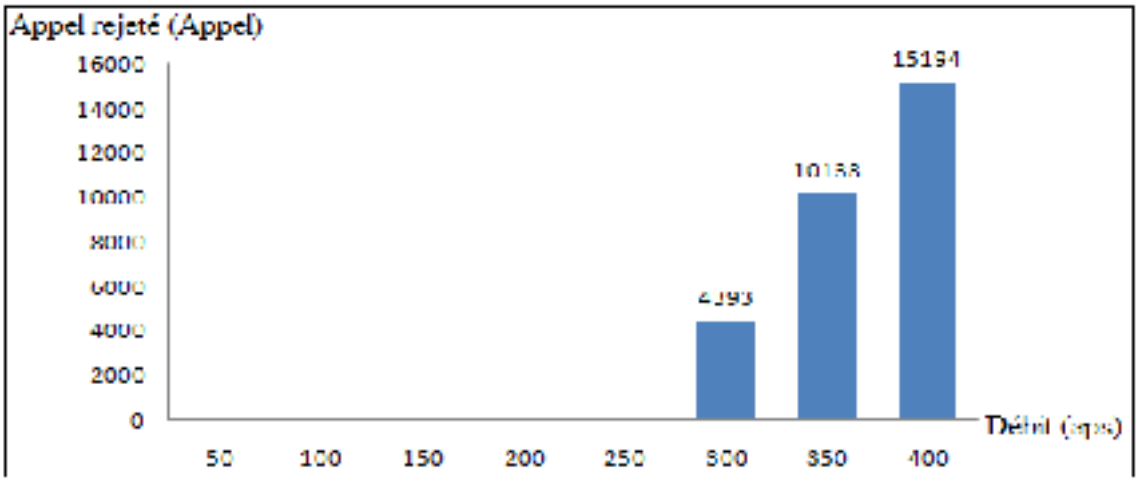

Figure 5.21 Le nombre d'appels rejeté

Durant les 5 premières rafales, le GFSIP était capable de traiter tous les messages sans aucun rejet. À partir de la 6eme (300 Appels/s), la capacité du système a atteint ses limites, et le rejet d'appel a commencé. Le GFSIP a reçu 170 100 appels (la somme de toutes les rafales), le nombre d'appels rejetés est de 29725 appels. Ceci dit que 83 % des appels ont été admis. Le fait de prioriser le message 'Invite' a permis au GFSIP d'accepter plus d'appels. Le délai de traitement du message 'Bye' et sa réponse '200 OK' a été réduit, ainsi, le GFSIP avait assez de ressources pour supporter plus d'appels.

Le fait de diminuer le délai d'attente du message 'Invite' implique que le taux de son admission sera élevé. Et donc, les facteurs qui ont contribué à minimiser le délai ont eu le même effet sur le Rejet.

Jusqu'à date, les résultats obtenus pour les trois systèmes sont différents. Une analyse comparative est donc nécessaire afin de conclure l'effet des disciplines sur les délais et les rejets.

5.7 FIFO-GFSIP vs FQ-GFSIP vs PQ-GFSIP

L'application des différentes disciplines a changé complètement le comportement du serveur SIP, ce qui a permis d'obtenir des résultats différents pour chacun des systèmes implémentés. Les résultats obtenus jusqu'à date ont démontré que la façon comment une file d'attente est servie, influent énormément sur les délais bout-en-bout et le taux de rejet. Les caractéristiques du trafic traité dans chaque discipline ont été déjà expliquées. À cet égard, une analyse comparative entre les déférents systèmes sera effectuée.

5.7.1 Les délais

Les figures 5.22 et 5.23 présentent l'évolution des délais selon les trois disciplines implémentées. La discipline FIFO paraît la moins efficace par rapport aux autres.

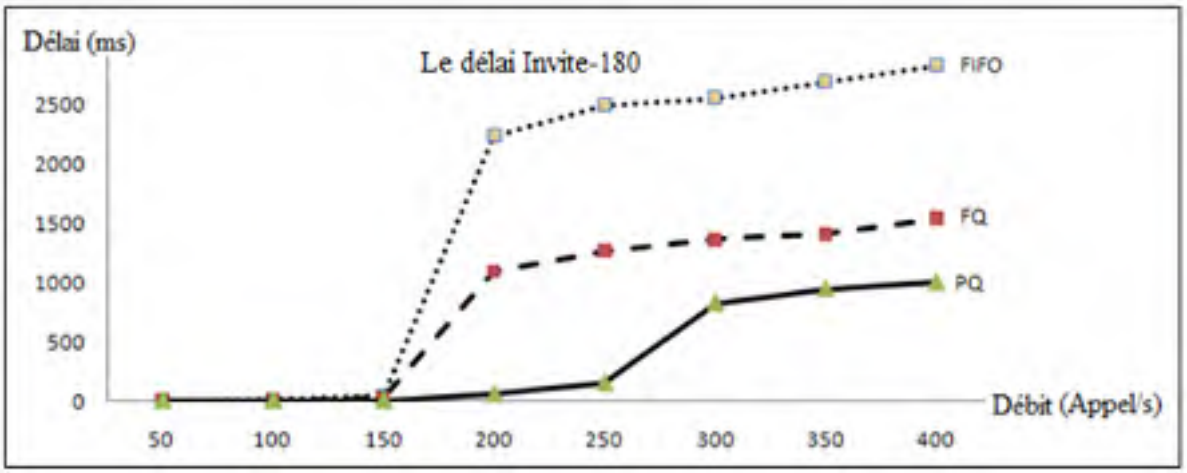

Figure 5.22 L'effet des disciplines sur le délai 'Invite-180'

En appliquant cette discipline, tous les messages qui arrivent pendant la rafale vont attendre la même durée. Ceci dit que les messages 'Invite' et '180 Ring' ont le même temps d'attente, d'où la valeur élevée du délai 'Invite-180'.

Les délais obtenus avec un FQ sont réduits par rapport à la discipline FIFO. Les délais 'Invite-180' sont restés dans une moyenne acceptable, tandis que les délais '200-Ack'

avaient une valeur beaucoup plus faible que celle obtenue en utilisant la discipline FIFO. L'avantage de cette discipline est qu'elle supporte les files d'attente longues contrairement au FIFO. Et donc, il sera possible de profiter au maximum du délai 'Invite-180' sans influencer le délai '200-Ack'. La discipline FQ profite de la dépendance explicite complète pour réduire le temps d'attente des messages '180 Ring' et 'Ack'. Une grande partie des délais bout-en-bout dépend seulement des messages 'Invite' et '200 OK'. C'est la raison pour laquelle les délais obtenus sont plus faibles que ceux du FIFO.

La discipline FIFO avait les résultats les plus faibles, car le délai des messages critiques était inacceptable notamment avec les files d'attente longues. Le FQ respecte la tolérance des messages 'Invite' ainsi que celle des messages critiques. En conséquence, cette discipline est considérée plus performante que FIFO au niveau des délais.

Il y avait des moments où la longueur de quelques files d'attente du FIFO-GFSIP dépasse la valeur maximale du nombre d'appels admis. Ce qui n'est pas le cas pour le FQ-GFSIP puisque les messages subséquents sont traités rapidement.

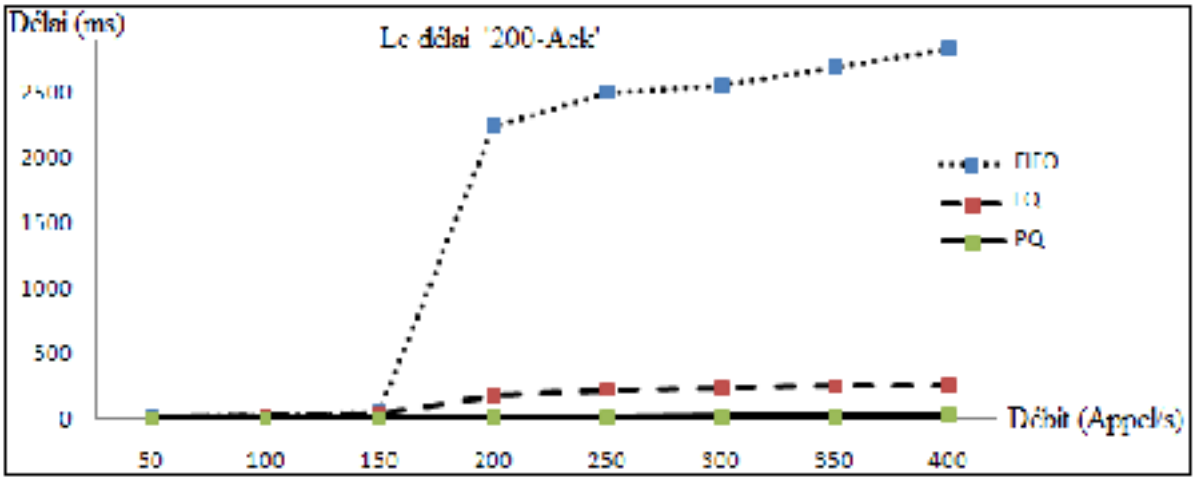

Figure 5.23 L'effet des disciplines sur le délai '200-Ack'

Le PQ a donné de meilleurs résultats pour les deux types de délai. La priorité des messages établie a réduit le délai d'attente des messages critiques. Ce qui a permis l'établissement de la session RTP le plus rapidement possible même dans une rafale de 400 appels/s. Le délai du message 'Invite' a, également, subi une diminution puisqu'il a été priorisé par rapport au message 'Bye' et sa réponse '200 OK'. Les FIFO-GFSIP et FQ-GFSIP perdent du temps dans le traitement de ces messages qui ne servent qu'à terminer les sessions.

Le but de l'implémentation d'un PQ-GFSIP est d'exploiter le temps de traitement de la signalisation de terminaison d'appel en faveur de la signalisation d'établissement d'appel. Naturellement, les rafales de trafic voix sont d'une durée courte et arrivent d'une façon inopinée. Il est donc possible de traiter les messages à basse priorité un peu plus tard.

Les résultats obtenus avec le PQ-GFSIP sont meilleurs par rapport aux deux autres systèmes. De plus, la performance restera intacte même si la file d'attente de 'Invite' est longue. Ceci, bien évidemment, augmentera le taux d'admission. En conséquence, le délai 'Invite-180' sera plus élevé. Toutefois, les délais des messages critiques seront toujours réduits, avec un minimum de taux de rejet. La longueur des files d'attente est un choix qui dépend de l'administrateur est la clientèle servie.

5.7.2 Les rejets

La figure 5.24 présente une comparaison entre les rejets effectués durant les trois différents tests.

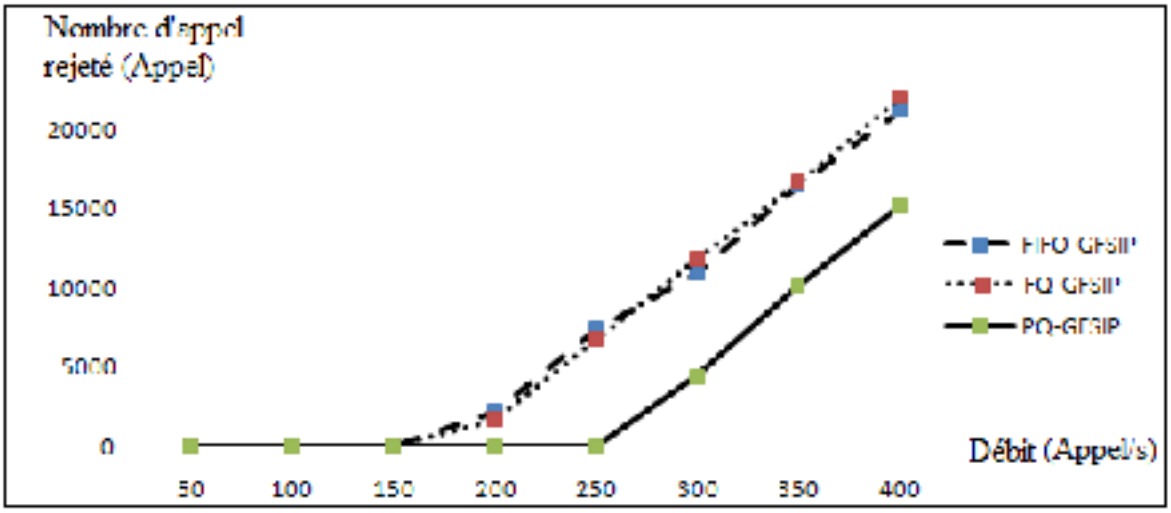

Figure 5.24 L'effet des disciplines sur le rejet

Le rejet d'appel est une opération très sensible au délai. Elle dépend de plusieurs facteurs, notamment le GC et l'utilisation du CPU et de la mémoire. L'admission d'un appel dépend de quelques millisecondes, et tout retard peut engendrer un nombre important de rejets. Pour chaque GFSIP, les tests ont été effectués plusieurs fois. Le nombre d'appels rejetés obtenu avait une valeur différente dans chacun des tests.

Le grand avantage de l'utilisation d'un PQ-GFSIP est la réduction du nombre d'appels rejetés, car seulement 17.48 % des appels ont été rejetés. En comparant cette valeur avec le pourcentage de rejet des deux autres systèmes, le réseau SIP a eu un gain de 16.93 % des appels qui ont été admis. Ceci se convertit en 28781 appels. Ce nombre d'appels a été rejeté lorsque le réseau a utilisé les deux systèmes : FIFO-GFSIP et FQ-GFSIP.

5.7.3 Conclusion

Les méthodes appliquées dans le traitement du message SIP par le GFSIP ont toutes contribué à la performance du système élaboré. La détection de retransmission, l'envoi du message '100 Trying' dès l'admission, ou la méthode de rejet implémentée à l'entrée du GFSIP, ont tous eu un effet considérable.

La contribution majeure consiste à combiner ces méthodes avec les trois disciplines vues précédemment. L'application de ces disciplines sur l'échange des messages a démontré un effet important sur la Qualité de service. Les trois disciplines appliquées avaient des effets différents sur les délais est les rejets. Chacune influence les délais et le taux de rejet d'une façon différente.

En résultat, le PQ était la discipline appropriée pour réduire les délais et le taux de rejet. Avec les résultats obtenus, il s'est avéré que la priorité des messages joue un rôle très important sur la gestion des messages SIP. La discipline FQ avait un effet très important sur les délais, mais non pas sur le rejet. La discipline FIFO est apparue avec la performance la plus faible vu les délais inacceptables des messages critiques.

CONCLUSION ET TRAVAIL FUTUR

Dans un réseau IP standard, les routeurs sont les éléments responsables sur l'acheminement des paquets. Leur congestion peut avoir un effet dramatique sur tout le réseau. Beaucoup d'effort a été mis pour résoudre ce type de problème parce qu'un routeur existe dans n'importe quel réseau de service IP.

Contrairement au routeur, le serveur SIP n'existe que dans un réseau de service SIP. Il est clair qu'ils n'ont pas le même degré d'importance, toutefois, il est possible de s'inspirer des solutions qui ont été proposées dans l'un pour les implémenter dans l'autre. Ceci est valable, parce que les deux éléments jouent, dans la plupart des cas, le même rôle. Un routeur achemine les paquets vers une destination, tandis qu'un serveur SIP achemine les messages vers les UAs/serveurs destinés. Toutefois, les deux éléments opèrent sur deux couches différentes.

Parmi les solutions qui sont implémentées dans les routeurs sont les disciplines des files d'attente. Ces disciplines ont eu un grand effet sur la qualité de service puisque les paquets n'attendent pas la même durée dans la file d'attente, ce qui permet au trafic sensible au délai d'être traité rapidement. Ainsi, le même principe peut être appliqué dans le cas d'un serveur SIP.

Naturellement un serveur SIP reçoit les messages et les met temporairement dans une file d'attente, le temps d'attente des messages dépends du temps du traitement et du débit de réception. Dans le cas d'un routeur, le trafic a des caractéristiques différentes, il y a ceux qui sont sensibles au délai d'autre sensible à la perte. Ce mémoire applique la même philosophie dans les serveurs SIP, il considère que les messages ont des caractéristiques différentes, il y a ceux qui sont sensibles au délai et le rejet, d'autre ont une petite tolérance au délai et rejet, d'autre sont tout à fait tolèrent.

Les résultats ont démontré que la différentiation entre les messages SIP, apporte une certaine amélioration à la qualité de service. Ceci confirme les résultats obtenus dans (Batteram, Meeuwissen et Bemmel, 2006). Le fait d'avoir des tolérances différentes dans la même session SIP, a permis d'établir une priorité pour chaque message. Le phénomène de dépendance entre messages SIP avait un effet variable sur les résultats en fonction de la discipline appliquée. En résultat : la signalisation de terminaison de l'appel devrait avoir la priorité la plus basse, ceci permettra de profiter de son temps de traitement pour augmenter le taux d'admission des appels et traiter les messages les plus prioritaires.

Ce travail se limite sur un seul service : celui de la voix. Une extension du travail peut s'effectuer afin d'investiguer les tolérances dans les autres services. Il est sera intéressant d'établir des classes de priorité de message dans tous les services qui utilisent SIP. Le serveur SIP devrait être en mesure de traiter les messages selon des classes bien définies. Les appels reçus peuvent avoir des priorités différentes. Cette approche peut être combinée avec les différents algorithmes de protection contre les congestions vues auparavant. Les algorithmes de '*Feedback*', peuvent inclure le principe de priorité des messages. Ceci permettra, bien évidemment, d'améliorer la qualité de service d'un réseau SIP d'une topologie Serveur-serveur.

La classe de trafic SIP peut être implémentée dans la couche réseau. Cette implémentation peut être combinée avec celle proposée dans ce mémoire. Les routeurs peuvent être configurés pour différencier entre les messages SIP et d'autre trafic. La congestion peut être cumulative, la rafale de trafic peut atteindre, dans ce cas, les routeurs et les serveurs SIP. L'étude peut être comparative et peut viser l'effet de la différentiation entre les messages SIP et le reste du trafic sur la tolérance.

RECOMMANDATIONS

Le traitement différencié des messages SIP est essentiel. La gestion des messages selon les tolérances devrait être inclue dans n'importe quel réseau SIP. Certes, ceci permettra d'avoir une certaine manipulation au niveau des délais. Il est, aussi, possible d'implémenter la méthode utilisée dans un serveur SIP, sans avoir recours à un GFSIP.

À cet égard, il est recommandé d'avoir des files d'attente dédiées pour chaque type de message, les types peuvent être établis selon l'implémentation et le besoin. Les files d'attente vont permettre le traitement les messages d'une façon adéquate. Cette méthode peut être utilisée pour respecter les contraintes de la qualité de service. L'élaboration des files d'attente devrait se faire d'une façon configurable, et doit être prise en considération dans la conception d'un serveur SIP. Outre, les disciplines appliquées sur cette file d'attente doivent aussi être présentes afin de permettre un certain choix dans la gestion des messages.

ANNEXE I

LE CODE DES MÉTHODES IMPLÉMENTÉES DANS LE SERVEUR SIP

```
/ *  la méthode qui traite les messages 'Invite'*/
protected void doInvite(SipServletRequest request) throws
IOException,ServletParseException,ServletException  {

                SipSession sipsession1 = request.getSession();
                SipSession sipsession2;
                SipServletRequest request2;
SipFactory sipFactory = (SipFactory) getServletContext().getAttribute(SIP_FACTORY);
                  if (sipFactory == null)
                   {
                      throw new ServletException("SipFactory Error ");
                    }
                      request2 = sipFactory.createRequest(request, true);
                      URI URIaddress =getIPaddress( request.getTo().getURI());
                      request2.setRequestURI(contactURI);
                      sipsession2 = request2.getSession();
                      sipsession1.setAttribute(LINK, sipsession2);
                      sipsession2.setAttribute(LINK, sipsession1);
                      sipsession1.setAttribute(USER, "caller");
                      sipsession2.setAttribute(USER, "callee");
              request2.setAttribute(LINK, req1);
              sipsession1.setAttribute("INVITE", request2);
               request2.send();
        }
```

```
/ *  Cette méthode  traite le message ‘180 Ring’ */
 protected void doProvisionalResponse(SipServletResponse resp)
                        throws ServletException, IOException
    {
                if (resp.getStatus() > 100)
         {
                SipServletRequest req1 = resp.getRequest();
                SipServletRequest req2 = (SipServletRequest)  req1.getAttribute(LINK);

                 int sc = resp1.getStatus();
                SipServletResponse resp2 = req2.createResponse(sc,
                resp1.getReasonPhrase());
                   try {
                        if (resp1.getContentType() != null) {
                        resp2.setContent(resp1.getRawContent(),resp1.getContentType());
                        }
                } catch (IOException ex) {}
                                resp2.send();
           }
      }

/ *  Cette méthode  traite les messages ‘200 OK’ de ‘Invite’ et ‘Bye’ */
   protected void doSuccessResponse(SipServletResponse response)
                        throws ServletException, IOException
         {
               if (response.getMethod().equals("INVITE")){
                  response.getSession().setAttribute(LINK, resp);
                 LongCallStart(response);
                }
             if (response.getMethod().equals("BYE")) {
                        response.getApplicationSession().invalidate();
            send(resp);
        }
```

```
/ *  Cette méthode  traite le message ‘Ack */
protected void doAck(SipServletRequest ack) throws ServletException,IOException
 {
    SipSession sipsession1 = ack.getSession();
     SipSession sipsession2 = (SipSession) sipsession1.getAttribute(LINK);
     if (sipsession2 == null) {
          ack.getApplicationSession().invalidate();
           return;
                          }
       SipServletResponse response;
      response = (SipServletResponse) sipsession2.getAttribute(LINK);
      SipServletRequest ack1 = response.createAck();
      try {
         if (ack.getContentType() != null) {
          ack1.setContent(ack.getRawContent(),ack.getContentType());
                           }
                  } catch (IOException ex) {}
      ack1.send();
         }
```

```
/ *  Cette méthode  traite le message ‘Bye’ */
protected void doBye( SipServletRequest request)
                        throws ServletException, IOException {
                LogcallEnd (request);
                SipSession sipsession1 = req1.getSession();
                SipSession sipsession2 = (SipSession) sipsession1.getAttribute(LINK);
                SipServletRequest request2 = sipsession2.createRequest("BYE");
                try {
                        if (request.getContentType() != null) {  request2.setContent(r
                        request.getRawContent(),request.getContentType());
                        }
                } catch (IOException ex) {}
                request2.setAttribute(LINK,request);
                request2.send();
        }
```

ANNEXE II

LE CODE DE LA CLASSE UTILISÉE POUR LES FILES D'ATTENTE

```
/*Les objets de cette classe présentent les file d'attente implémentés pour chacun des
messages SIP dans le GFSIP */

import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import javax.servlet.sip.SipServletMessage;
public class Fifo {

   private int maxSize;
   public  int QueueFrontindex=0;
   public  int QueueEndindex=0;
 private SipServletMessage[] queArray;

 public Fifo(int s) {
   maxSize = s;
   queArray = new SipServletMessage[maxSize];
 }
 public void insert(SipServletRequest req) {

     queArray[QueueEndindex] = req; // insert it
     QueueEndindex++;
   } // end else (nItems > 0)
 public void insert(SipServletResponse res) {
     queArray[QueueEndindex] = res;
     QueueEndindex++;
   }
```

```
 public SipServletMessage remove(){
    SipServletMessage Temp= queArray[QueueFrontindex];
    queArray[QueueFrontindex]=null;
  Runtime.getRuntime().freeMemory();
    QueueFrontindex++;
   return Temp;
 }
 public boolean isRequest(int i)
   {
    boolean state=false;
    try{
    if ((SipServletRequest)queArray[i]!=null)
        state= true;
       }
    catch(ClassCastException e)
    { state=false;
    }
    return state;
 }
public SipServletMessage display(int i){
   return queArray[i];
   }
 public boolean isEmpty(){
   return (QueueEndindex == QueueFrontindex);
 }
 public void setEmpty()
   {
 QueueEndindex = QueueFrontindex=0;
 queArray = new SipServletMessage[10];
```

```
  }
public int getQueueLenght()
      {
int i= QueueEndindex - QueueFrontindex;
return i;
}
}
```

ANNEXE III

LE CODE DU CLASSIFICATEUR

```
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;

public class Classificateur {
  Fifo Queue0 ;
  Fifo Queue1 ;
  Fifo Queue2 ;
  Fifo Queue3 ;
  Fifo Queue4 ;
  Fifo Queue5 ;
public Classificateur(Fifo Queue0,Fifo Queue1,Fifo Queue2,Fifo Queue3,Fifo Queue4,Fifo
Queue5 )
    {
    this.Queue0=Queue0;
    this.Queue1=Queue1;
    this.Queue2=Queue2;
    this.Queue3=Queue3;
    this.Queue4=Queue4;
    this.Queue5=Queue5;
   }
public void classifier(SipServletRequest req,VoiceProxyFQ proxy)
    {
    if(req.getMethod().equals("INVITE"))
       Queue0.insert(req);
      if(req.getMethod().equals("ACK"))
      Queue3.insert(req);
```

```
    if(req.getMethod().equals("BYE"))
    Queue4.insert(req);
}
public void classifier(SipServletResponse resp,VoiceProxyFQ proxy)
   {
    if(resp.getStatus()==180)
    Queue1.insert(resp);
    if(resp.getStatus()==200 && resp.getMethod().equals("INVITE") )
    Queue2.insert(resp);
    if(resp.getStatus()==200 && resp.getMethod().equals("BYE") )
    Queue5.insert(resp);
}
}
```

ANNEXE IV

LE CODE DE L'ORDONNANCEUR AVEC LA DISCIPLINE FQ

```
 /*le code de la discipline FQ*/

import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import java.util.Timer;
import java.util.TimerTask;

public class schedulerFQ {
    public SelectorTask task;
    public Classificateur classificateur;
    public static VoiceProxyFQ proxy;
    public Fifo QueueInvite = new Fifo(100000) ;
    public Fifo Queue180 =new Fifo(100000) ;
    public Fifo Queue200 =new Fifo(100000) ;
    public Fifo QueueAck =new Fifo(100000) ;
    public Fifo ByeQueue =new Fifo(100000) ;
    public Fifo Bye200Queue =new Fifo(100000) ;

public  PriorityMessageManager(VoiceProxyFQ proxy){
      this.proxy = proxy;
      SelectorTask task= new SelectorTask();
     task.setProxy(this,proxy);
     task.setDaemon(true);
     task.start();
     classificateur = new +
Classificateur(QueueInvite,Queue180,Queue200,QueueAck,ByeQueue,Bye200Queue);
```

```
}
public void ProcessFQ()
   {
while(!QueueInvite.isEmpty()||!Queue180.isEmpty()||!Queue200.isEmpty()||!QueueAck.isE
mpty()||!ByeQueue.isEmpty()||!Bye200Queue.isEmpty())
      {
        ProcessQueueInvite();
        ProcessQueue180();
        ProcessQueue200();
        ProcessQueueAck();
        ProcessByeQueue();
        ProcessBye200Queue();
      }
}
public  void ProcessQueueInvite() {
  if(!QueueInvite.isEmpty())
  {
     SipServletRequest request=  (SipServletRequest)QueueInvite.remove();
       try{
      send(request);
       }catch (Exception e){}
           }
   }
public  void ProcessQueue180() {
  if(!Queue180.isEmpty())
     {SipServletResponse resp=(SipServletResponse)Queue180.remove();
       try{
          send (resp);
        }catch (Exception e){}
           }
```

```
        }
public  void ProcessQueue200() {
  if(!Queue200.isEmpty())
     {
     SipServletResponse resp=(SipServletResponse)Queue200.remove();
      try{
         send (resp);
        }catch (Exception e){}
       }
   }
public  void ProcessQueueAck()
 {
  if(!QueueAck.isEmpty())
     {SipServletRequest Ack=(SipServletRequest)QueueAck.remove();
      try{
         send (Ack);
           }catch (Exception e){}
      }
 }
public  void ProcessByeQueue() {
  if(!ByeQueue.isEmpty())
     {
     try{
        if (ByeQueue.isRequest(ByeQueue.QueueFrontindex)){
       SipServletRequest request=(SipServletRequest)ByeQueue.remove();
      send (request);
          }
        }catch (Exception e){}
   }
   }
```

```
public  void ProcessBye200Queue() {
if(!Bye200Queue.isEmpty())
    {
      try{
       SipServletResponse resp=(SipServletResponse)Bye200Queue.remove();
       send(resp);
       }
      catch (Exception e){}
 }
}
```

ANNEXE V

LE CODE DE L'ORDONNANCEUR AVEC LA DISCIPLINE PQ

```
/*le code de la discipline PQ*/

import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import java.util.Timer;
import java.util.TimerTask;

   public class PriorityMessageManager {
   public SelectorTask task;
   public Classificateur classificateur;
   public static VoiceProxyPQ proxy;
   public Fifo QueueInvite = new Fifo(100000) ;
   public Fifo Queue180 =new Fifo(100000) ;
   public Fifo Queue200 =new Fifo(100000) ;
   public Fifo QueueAck =new Fifo(100000) ;
   public Fifo ByeQueue =new Fifo(100000) ;
   public Fifo Bye200Queue =new Fifo(100000) ;


public  PriorityMessageManager(VoiceProxyPQ proxy){

    this.proxy = proxy;
    task= new SelectorTask();
    task.setProxy(this,proxy);
    task.setDaemon(true);
    task.start();
```

```
    classificateur = new
Classificateur(QueueInvite,Queue180,Queue200,QueueAck,ByeQueue,Bye200Queue);
}
public void ProcessPQ()
   {
while(!QueueInvite.isEmpty()||!Queue180.isEmpty()||!Queue200.isEmpty()||!QueueAck.isE
mpty()||!ByeQueue.isEmpty()||!Bye200Queue.isEmpty())
      {
   ProcessQueueInvite();
   ProcessBye200Queue();
       }
 }
public void ProcessQueueInvite()
{
while(!QueueInvite.isEmpty()||!Queue180.isEmpty()||!Queue200.isEmpty()||!QueueAck.isE
mpty() )
  {
     ProcessQueue180();
    if(!QueueInvite.isEmpty())
     {
     SipServletRequest request=  (SipServletRequest)QueueInvite.remove();
       try{
      send(request);
       }catch (Exception e){}
     }
  }
   }
```

```
public  void ProcessQueue180() {
  while(!Queue180.isEmpty()||!Queue200.isEmpty()||!QueueAck.isEmpty() )
     {
      ProcessQueue200();
      if(!Queue180.isEmpty())
      {
     SipServletResponse resp=(SipServletResponse)Queue180.remove();
       try{
          send(resp);
          }catch (Exception e){}
        }
    }
  }
public  void ProcessQueue200() {
   while(!Queue200.isEmpty() || !QueueAck.isEmpty())
  {
     ProcessQueueAck();
     if(!Queue200.isEmpty())
     {
     SipServletResponse resp=(SipServletResponse)Queue200.remove();
      try{
     send(resp);
        }catch (Exception e){}
     }
  }
}
public  void ProcessQueueAck() {
  if(!QueueAck.isEmpty())
     {
```

```
      SipServletRequest Ack=(SipServletRequest)QueueAck.remove();
       try{
      send(Ack);
         }catch (Exception e){}
      }
 }
public  void ProcessByeQueue() {
  while(!ByeQueue.isEmpty())
      {
      try{
         if (!ByeQueue.isEmpty())
         {
        SipServletRequest request=(SipServletRequest)ByeQueue.remove();
       send(request);
          }
         }catch (Exception e){}
      ProcessQueueInvite();
  }
   }
  public  void ProcessBye200Queue() {
       ProcessByeQueue();
        while (!Bye200Queue.isEmpty())
        { ProcessByeQueue();
        try{
         SipServletResponse resp=(SipServletResponse)Bye200Queue.remove();
         send(resp);
         }
        catch (Exception e){}  }
}
}
```

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Aziz, W. A., S. H. Elramly et M. M. Ibrahim. 2010. « Capacity Measurement of IP–Multimedia Subsystem (IMS) Controllers ». Dans *Computational Intelligence, Modelling and Simulation (CIMSiM), la deuxiéme conférence internationale le* 28-30 Sept. 2010. p.p. 541-545.

Batteram, Harold, Erik Meeuwissen et Jeroen van Bemmel. 2006. « SIP message prioritization and its applications ». *Bell Labs Technical Journal,* vol. 11, n^{o} 1, p.p. 21-36.

Free Software Foundation, Inc. 2011. *GNU Operating System*. En ligne. <http://www.gnu.org/software/ncurses/ncurses.html>. Consulté le 15 Mai 2011.

Gao, Zhiguo, Zhe Xiang, Wei Xue, Zhiyong Liang et Bo Yang. 2009. « SIP Offload Engine for Accelerating J2EE Based SIP Application Server ». Dans *Communication Software and Networks, 2009. ICCSN '09. Conférence internationale le* 27-28 Fev. 2009. p.p. 749-753.

Henry Sinnreich, Alan B. Johnston 2001. *Internet Communications Using SIP* Coll.
« Delivering VoIP and Multimedia Services with Session Initiation Protocol ». N.Y: John Wiley & Sons, Inc., 298 p.

Hewlett-Packard. 2004. *SIPp*. En ligne. <http://sipp.sourceforge.net/ >. Consulté le 30 Avril 2011.

Hilt, V., et I. Widjaja. 2008. « Controlling overload in networks of SIP servers ». In *Network Protocols, 2008. ICNP 2008. IEEE Conférence internationale le* 19-22 Oct. 2008 . p.p. 83-93.

Hrischuk, C. . 2006. «A Tutorial on SIP Application Server Performance and Benchmarking». *CMG -CONFERENCE-,* vol. 2, p.p. 729-740.

iptel. 2011. *iptel.org*. En ligne. <http://www.iptel.org/sip/intro/messages >. Consulté le 1er Mai 2011.

J.Rosenberg, H.S., G.Camarillo, A.Jahnston. 2002. « RFC 3261 SIP : Session Initation Protocol ».

Jing, Sun, Yu Hongliang et Zheng Weimin. 2008. « Flow Management with Service Differentiation for SIP Application Servers ». Dans la conférence annuelle *ChinaGrid, 2008. le* 20-22 Août. 2008. p.p. 272-277.

Jing, Sun, Hu Jinfeng, Tian Ruixiong et Yang Bo. 2007. « Flow Management for SIP Application Servers ». In *Communications, 2007. ICC '07. IEEE International Conference on* (24-28 Juin 2007). p.p. 646-652.

Jing, Sun, Tian Ruixiong, Hu Jinfeng et Yang Bo. 2009. « Rate-based SIP flow management for SLA satisfaction ». In *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on* (1-5 Juin 2009). p.p. 125-128.

Kadoch, Michel. 2008a. *MGR840 Mobilité et téléphonie IP: notes du cours MGR840.* Programme de maîtrise en génie avec concentration : réseaux de télécommunications. Montréal: École de technologie supérieure.

Kadoch, Michel. 2008b. *MGR-820: Réseau haut débit et nouvelles technologies de IP: notes de cours MGR-820.* Programme de maîtrise en génie avec concentration : réseaux de télécommunications. Montréal: École de technologie supérieure.

M. Day, J. Rosenberg, H. Sugano. 2000. « RFC 2778: A Model for Presence and Instant Messaging ».

Montagna, S., et M. Pignolo. 2008. « Performance Evaluation of Load Control Techniques in SIP Signaling Servers ». Dans *Systems, 2008. ICONS 08. troisiéme conférence international. Le* 13-18 April 2008. p.p. 51-56.

Oracle. 2011a. *Communication and Mobility Server Developer's Guide*. En ligne. <http://download.oracle.com/docs/cd/E10301_01/doc.1013/e10293/sipserv.htm>. Consulté le 20 Mai 2011.

Oracle. 2011b. *JSR 116: SIP Servlet API*. En ligne. <http://www.jcp.org/en/jsr/detail?id=116>. Consulté le 15 Mai 2011.

Oracle. 2011c. *JSR 289: SIP Servlet API*. En ligne. <http://www.jcp.org/en/jsr/detail?id=289>. Consulté le 15 Mai 2011.

Oracle. 2011d. *Netbeans*. En ligne. <http://netbeans.org/>. Consulté le 1er Septembre 2010.

Shen, Charles, Henning Schulzrinne et Erich Nahum. 2008. « Session Initiation Protocol (SIP) Server Overload Control: Design and Evaluation ». In *Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, sous la dir. de Schulzrinne, Henning, Radu State et Saverio Niccolini. Vol. 5310, p.p. 149-173. Coll. « Lecture Notes in Computer Science ». Springer Berlin / Heidelberg. <http://dx.doi.org/10.1007/978-3-540-89054-6_8>.

Sisalem, Dorgham. 2011. *iptel.org: SIP Overload Control: Where are we today?* En ligne. <www.iptel.org/~dor/papers/Sisa1009:SIP.pdf >. Consulté le 10 Avril 2011.

Sourceforge. 2010. *TCPDUMP & LIBCAP*. En ligne. <http://www.tcpdump.org/>. Consulté le 10 Mai 2011.

V. Hilt, H. Schulzrinne. 2010. « Session Initiation Protocol (SIP) Overload Control draft-hilt-sipping-overload-08 ».

Van Den Bossche, B., F. De Turck, B. Dhoedt, P. Demeester, G. Maas, J. Moreels, B. Van Vlerken et T. Pollet. 2006. « ISE01-2: J2EE-based Middleware for Low Latency Service Enabling Platforms ». Dans *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE* (27 Nov 2006 - 1 Dec 2006). p.p. 1-6.

Chris Boulton, Kristoffer Gronowski et Inc Books24x. 2009. *Understanding SIP Servlets 1.1* 312 p.

Erich Nahum, Fred Minger, Erich Nahum, Zhijian (Ed) Pan, Xiping Wang, Charles Wright. 2007. *SIP Application Server Benchmark*. New York: IBM T. J. Watson Research Center, 33 p.