

TABLE DES MATIÈRES

	Page
INTRODUCTION	1
CHAPITRE 1 LES CARACTÉRISTIQUES DE TRANSMISSION MÉDIA SUR LE RÉSEAU INTERNET	7
1.1 Réseau Internet.....	7
1.1.1 L'hétérogénéité du réseau Internet.....	7
1.1.2 Les conditions variables du réseau Internet.....	8
1.2 Qualité de service.....	8
1.3 Transport du média TCP ou UDP et pour quel motif.....	9
1.4 Le protocole temps réel RTP	10
1.4.1 Format d'un paquet RTP.....	11
1.4.2 Architecture d'intégration du module RTP.....	12
1.5 Les codecs audio les plus utilisés en transmission média.....	14
1.6 Codecs vidéo.....	15
1.7 Conclusion	16
CHAPITRE 2 SIP, LE PROTOCOLE DE SIGNALISATION MULTIMÉDIA.....	17
2.1 L'objectif de développement du protocole SIP	17
2.2 Établissement d'une session	18
2.2.1 Architecture SIP.....	18
2.2.2 Méthodes.....	19
2.2.3 Réponses de requêtes	20
2.2.4 Format de messages	20
2.2.5 L'enregistrement du client SIP	21
2.2.6 Les messages de la signalisation.....	22
2.3 Conclusion	23
CHAPITRE 3 IAX, LE PROTOCOLE DE TRANSMISSION MÉDIA SUR LA PLATEFORME IPPBX	24
3.1 L'objectif de conception	24
3.2 Appel basique IAX	25
3.2.1 La séquence de messages d'enregistrement.....	25
3.2.2 La signalisation de terminaux IAX.....	27
3.2.3 Mini-Frame	29
3.3 Conclusion	30
CHAPITRE 4 ENVIRONNEMENT EXPÉRIMENTAL ET RECHERCHES ANTÉRIEURES	31
4.1 Les exigences de l'environnement expérimental.....	31
4.2 Les éléments de l'architecture testés et les alternatives.....	32

4.3	Contexte de l'environnement expérimental et	34
4.4	Description de la plateforme IPPBX et les recherches associées	36
4.5	Installation de l'IPPBX Asterisk.....	39
4.5.1	Matériels utilisés	39
4.5.2	Script du déploiement	40
4.5.3	Création d'extensions SIP.....	40
4.5.4	Création d'extensions IAX	41
4.5.5	Groupes et manipulations des appels.....	41
4.6	Interconnexion au réseau PSTN.....	42
4.7	Interconnexion au réseau GSM.....	43
4.8	Interconnexion de deux systèmes IPPBX.....	44
4.9	Conclusion	46
CHAPITRE 5 SIMULATION : SIGNALISATION (SIP/IAX) ET LA QUALITÉ DE LA COMMUNICATION IAX.....		47
5.1	Description de la simulation calcul de signalisation SIP/IAX.....	47
5.2	La performance d'un canal IAX avec le codec G.711	49
5.2.1	La méthodologie appliquée pour calculer la bande passante.....	50
5.2.2	La méthodologie appliquée pour calculer le délai inter arrivée.....	51
5.2.3	La méthodologie appliquée pour calculer la gigue.....	52
5.3	La performance du canal IAX avec le codec GSM	53
5.3.1	La bande passante GSM.....	54
5.3.2	Le délai inter-arrivée GSM	55
5.3.3	La gigue GSM.....	55
5.4	Les événements IAX.....	56
5.5	Conclusion	57
CHAPITRE 6 IMPLÉMENTATION DU PROTOCOLE RTP DANS UN CLIENT MOBILE DE VIDÉOCONFÉRENCE		59
6.1	Développement d'un client de vidéoconférence mobile.....	59
6.1.1	La technologie utilisée J2ME.....	61
6.1.2	La solution proposée	62
6.1.3	JSR SIP	62
6.1.4	JSR 135 Mobile MultiMedia	64
6.2	Les paquets RTP	66
6.3	Les caractéristiques supportées du client vidéoconférence mobile	68
6.4	Résultats de l'implémentation : vidéoconférence mobile.....	74
6.5	Discussion de la partie applicative.....	77
6.6	Conclusion	78
CONCLUSION.....		79
RECOMMANDATIONS		81
ANNEXE I SCRIPT DE COMPILATION LA PLATEFORME IPPBX		82

ANNEXE II CONFIGURATION DES EXTENSIONS SIP.....	84
ANNEXE III CONFIGURATION DES EXTENSIONS IAX.....	85
ANNEXE IV CONFIGURATION DU PLAN DE NUMÉROTATION	86
ANNEXE V MODULE DU CALCUL BANDE PASSANTE IAX	87
ANNEXE VI MODULE DU CALCUL DÉLAI IAX.....	90
ANNEXE VII MODULE DU CALCUL GIGUE	92
ANNEXE VIII L'ENREGISTREMENT DU CLIENT MOBILE SIP.....	94
ANNEXE IX LA SIGNALISATION DU CLIENT MOBILE SIP.....	100
ANNEXE X L'IMPLÉMENTATION DU PROTOCOLE RTP	117
LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES.....	120

LISTE DES TABLEAUX

	Page
Tableau 1.1	Le type de contenu pour l'encodage audio14
Tableau 1.2	Les codecs d'encodage vidéo16
Tableau 4.1	La qualité de l'appel en utilisant MOS (Mean Opinion Score)37
Tableau 5.1	Les statistiques d'enregistrement et de signalisation SIP/IAX.....48
Tableau 5.2	Les événements IAX.....57
Tableau 5.3	La récapitulation de communication avec les codecs G.711 et GSM.....58

LISTE DES FIGURES

	Page
Figure 1.1	Le format d'un paquet RTP.11
Figure 1.2	L'entête d'un paquet RTP.12
Figure 1.3	La composition d'un paquet RTP.13
Figure 2.1	L'architecture du protocole SIP.....18
Figure 2.2	Les séquences d'enregistrement (le délai Setup).22
Figure 2.3	Le diagramme de signalisation SIP (le délai Setup).22
Figure 3.1	Les requêtes d'enregistrement IAX.26
Figure 3.2	Les séquences d'enregistrement IAX pour calculer le délai Setup.27
Figure 3.3	Les séquences de signalisation IAX.28
Figure 4.1	L'architecture de composants techniques.34
Figure 5.1	La bande passante estimée dans les deux sens.....51
Figure 5.2	Le délai inter arrivée G.711 (ms).52
Figure 5.3	La gigue en ms.53
Figure 5.4	La bande passante du canal IAX avec le codec GSM.54
Figure 5.5	Le délai inter arrivée d'une communication IAX avec le codec GSM (ms).55
Figure 5.6	La gigue d'une communication IAX avec le codec GSM (ms).55
Figure 5.7	Les messages envoyés au cours d'une communication IAX.....56
Figure 6.1	Les transactions de requêtes : mobile, proxy et soft phone.63
Figure 6.2	Les composants multimédias supportés par l'émulateur mobile.....65
Figure 6.3	L'enregistrement de l'émulateur mobile auprès du Proxy/Registrar.69
Figure 6.4	L'enregistrement de l'émulateur mobile auprès du proxy SIP (J2ME).70

Figure 6.5	Le contenu du protocole SDP.	71
Figure 6.6	L'établissement de la communication vidéoconférence.....	72
Figure 6.7	Le flux audio.	73
Figure 6.8	Les statistiques du protocole RTP.....	74
Figure 6.9	Le flux vidéo.	74
Figure 6.10	La manipulation du flux audio et vidéo en mode binaire.	75
Figure 6.11	La lecture du flux audio.	75
Figure 6.12	La lecture du flux vidéo.	76
Figure 6.13	La capacité de la mémoire utilisée pendant la session vidéoconférence. ..	77

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ACK	Acknowledge
ADSL	Asymmetric Digital Subscriber Line
ATM	Asynchronous Transfer Mode
CIF	Common Intermediary Format
DCT	Discrete Cosine Transform
DNS	Domain Name System
DS	Digital Signal
FCS	Frame Check Sequence
FTP	File Transfer Protocol
HF	Hybrid Fiber Coaxial
HTTP	Hyper Text Transfer Protocol
IAX	Inter Asterisk Exchange
IEEE	Institute for Electrical and Electronic Engineer
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
ISDN	Integrated Services Digital Network
LAN	Local Area Network
MAC	Media Access Control
MBZ	Must Be Zero
MD5	Message Digest
MGCP	Media Gateway Control Protocol

NTP Network Time Protocol

OUI Organizationally Unique Identifier

PCM Pulse Code Modulation

PSTN Public Switched Telephone Networks

QCIF Quarter Common Intermediary Format

RFC Request For Comment

RGB Red Green Blue

RSA Rivest Shamir Adleman

RSVP Resource Reservation Protocol

RTCP Real Time Control Protocol

RTP Real Time Protocol

SAP Session Announce Protocol

SDP Session Description Protocol

SIP Session Initiation Protocol

SMTP Simple Mail Transfer Protocol

SQCIF Sub-Quarter Common Intermediary Format

SSRC Synchronization Source

TCP Transmission Control Protocol

TOS Type Of Service

UAC User Agent Client

UAS User Agent Server

UDP User Datagram Protocol

UIT Union International Telecommunication

WFQ Weighted Fair Queuing

INTRODUCTION

Les progrès technologiques du réseau Internet ont permis l'innovation de nouveaux services sur les réseaux IP. Les applications voix sur IP, vidéo et vidéoconférence sont devenues des domaines importants de recherche et de développement pour l'industrie des télécommunications. La demande pour les technologies de communication audiovisuelle augmentera encore dans les prochaines années [31]. Les opérateurs de télécommunication déploient de nouveaux services afin de rentabiliser les infrastructures mises en place (ADSL, UMTS et IMS). Les constructeurs de terminaux mobiles et fixes élaborent de nouveaux systèmes pour permettre une communication audiovisuelle universelle. De nouveaux standards sont en cours pour unifier la transmission multimédia sur les réseaux IP sans se soucier de l'architecture matérielle ou logicielle du support de transmission. Les grands acteurs de l'industrie multimédia souhaitent vendre leur contenu en ligne sous forme de bibliothèque multimédia numérique [31].

Les applications de communication audiovisuelle peuvent apporter une valeur ajoutée aux entreprises en particulier et à l'humanité en général. La communication multimédia peut réduire le coût et le temps de déplacement des employés appartenant à une entreprise multinationale. Les ambulanciers peuvent recevoir des indications d'urgences par la vidéoconférence pour traiter un patient. Les zones isolées géographiquement peuvent également communiquer visuellement avec un centre hospitalier pour recevoir les services médicaux nécessaires. Enfin, la communication visuelle permet aux familles éloignées de garder un contact visuel et réel malgré la distance qui les sépare. Tous les éléments mentionnés précédemment ont motivé et encouragé le développement des applications multimédia avec l'utilisation du standard de communication.

Les protocoles de signalisation SIP et IAX peuvent être exécutés sur un client, un terminal mobile ou un serveur. Les requêtes d'enregistrement et de signalisation peuvent présenter des délais significatifs sur les liaisons à bas débit, surtout si le client doit retransmettre ces requêtes à plusieurs reprises. Les délais peuvent avoir un impact important au cours d'une

session multimédia. La transmission du flux de voix ou vidéo en temps réel peut être retardée et affectée menant à une qualité de service qui n'est plus adéquate.

Ces dernières années ont été remarquables par la mise en œuvre de terminaux mobiles, l'utilisation de ces appareils est quotidienne, surtout plusieurs standards et systèmes d'encodage et de traitement multimédia ont été conçus spécifiquement pour les terminaux mobiles. Toutefois, on constate que la généralisation du transport voix et vidéo est loin d'être fixe et continue d'évoluer.

Le protocole SIP utilise des ports séparés pour transmettre les données de signalisation et les données média. Ce concept pose un problème sur les réseaux équipés NAT (Network Address Translation). Le client SIP choisit un port dynamique pour recevoir le flux média. Ce port n'est pas reconnu sur le routeur, car le client SIP est associé au port établi pendant la connexion.

Pour résoudre ce problème, le récent protocole IAX utilise un port unique pour la signalisation et le transport des données média. Le protocole de signalisation et de transport média IAX, qui se trouve actuellement sous forme draft [34], est en voie de normalisation et offre d'autres nouvelles fonctionnalités intéressantes. Il permet par exemple de transporter les données média dans un format binaire compact offrant un débit optimisé. Les différentes structures de trames IAX permettront d'optimiser le transfert des requêtes de signalisation, les données brutes médias, et la combinaison de différents flux sur le même lien [34].

Il existe d'autres différences notables entre SIP et IAX. Lorsque la signalisation est effectuée via le protocole SIP, le transport média nécessite 12 octets pour l'en-tête du protocole RTP alors que la signalisation IAX nécessitera 4 octets d'en-tête pour transmettre les données voix ou vidéo.

Le protocole IAX étant récent, il n'existe aucune étude qui traite de ses performances avec le codec G.711 supporté sur le réseau téléphonique traditionnel et le codec GSM utilisé sur le réseau cellulaire.

Ce constat nous étonne et nous pousse à analyser les deux protocoles sur le plan théorique expérimental. Nous constatons aussi qu'il n'existe même pas de plateforme expérimentale permettant de mener une telle étude. En effet, il n'existe pas d'architecture expérimentale globale capable de commuter les canaux SIP et IAX tout en offrant la possibilité d'adhérer différents types de clients (client SIP, client IAX et terminal mobile). Une plateforme expérimentale devra donc être conçue et réalisée. De plus, il n'est pas clair quels outils seront adéquat afin d'être inclus dans cette plateforme pour mesurer le temps requis d'enregistrement de signalisation SIP/IAX et aussi les performances de canaux IAX.

Finalement, nous avons constaté que le protocole RTP n'était pas disponible sur les émulateurs de terminaux mobiles. Cette composante est cruciale à la plateforme expérimentale pour permettre le transport de flux audiovisuels dans un environnement de vidéoconférence mobile.

Cette recherche permettra de tester et analyser le protocole IAX pour mieux comprendre et valider les avantages du protocole IAX par rapport à SIP au niveau conceptuel ainsi que les gains possibles au niveau de la bande passante. L'utilité de ce travail est de découvrir et valider les avantages au plan théorique et expérimental du nouveau protocole IAX que l'on entend souvent dans le jargon VoIP (Voice Over Internet Protocol).

L'intérêt d'implémenter le protocole RTP dans l'émulateur de terminaux mobiles selon RFC 3550 est de pouvoir tester les services de réseaux de nouvelle génération. Nous croyons également que l'architecture expérimentale pourra unifier différentes technologies.

1. Description du contexte et objectif du travail

Les protocoles de transmission média sont nombreux du point de vue du modèle OSI. Dans ce mémoire, le travail consiste à étudier et intégrer les protocoles de transmission média au niveau de la couche applicative et plus spécifiquement le protocole SIP, RTP et IAX.

La transmission média exige la signalisation pour permettre l'établissement d'une session de communication audiovisuelle, par l'intermédiaire d'un protocole comme Session Initiation Protocole (SIP), qui négocie les paramètres de connexion, présence et disponibilité. Ce dernier fait appel à un autre protocole Session Description Protocol (SDP) pour négocier les paramètres de médias audiovisuels, lorsque les paramètres de signalisation, connexion, et média sont déterminés, la partie communicante fait appel au protocole de transfert média en temps réel Real Time Protocol (RTP) pour acheminer l'audio ou la vidéo au destinataire.

Le protocole IAX effectue la signalisation et la transmission média par l'intermédiaire de différentes structures du protocole. Ce projet propose une architecture expérimentale pour tester et analyser les performances de SIP et IAX. Ce travail se charge de calculer le délai requis pour l'enregistrement et la signalisation (SIP et IAX), effectuer un test de performance des communications audio transportées via le protocole IAX, et intégrer le protocole de transmission média en temps réel (RTP) dans les terminaux mobiles.

2. Contributions du mémoire

Ce mémoire apporte quatre contributions majeures :

1. Une architecture expérimentale de test capable de commuter les canaux SIP et IAX et fournir les outils appropriés pour calculer le délai signalisation et la performance. Nous croyons que cette conception architecturale est innovatrice au terme des moyens offerts pour tester différents aspects de transmission média (réseau, interconnexion avec le réseau PSTN, interconnexion avec le réseau GSM, la sécurité réseau, et l'encodage médias)

2. L'installation de la plateforme IPPBX pour calculer le temps requis pour la signalisation en utilisant la même procédure discutée dans [2] et [3].
3. Le test de performances de communication audio via le protocole IAX.
4. L'analyse du protocole RTP est effectuée pour implémenter RTP dans les émulateurs de terminaux mobiles, et pour permettre la simulation d'un client de vidéoconférence mobile via les protocoles SIP, SDP et RTP. Ensuite nous allons tester les performances associées (la bande passante, la gigue et la taille de mémoire). L'implémentation du protocole RTP dans les terminaux mobiles se fait via la technologie J2ME et la plateforme IPPBX. Cela représente une nouvelle approche d'implémentation du protocole de transmission média en temps réel dans les terminaux mobiles, basée sur le travail de [8] mais utilise une nouvelle conception d'intégration de paquets RTP dans les datagrammes User Datagram Protocol (UDP).

3. Structure du mémoire

Le mémoire se compose de six chapitres. La transmission de médias audiovisuels est étudiée en détail dans chaque couche en commençant par la capture du média, en passant par les différentes opérations d'encodage, construction de paquets, transmission et l'arrivée au destinataire.

- Le chapitre 1 décrit l'hétérogénéité du réseau Internet meilleur-effort et son impact sur la qualité de service des applications multimédia. Ensuite le chapitre se focalise sur le transport et la transmission des médias. Les protocoles de transmission Transmission Control Protocol (TCP) et User Datagram Protocol (UDP) sont requis pour n'importe quel transfert. On aborde les applications nécessitant TCP ainsi que celles nécessitant UDP en présentant les motifs. Enfin, on présente les protocoles RTP et RTSP qui ont été conçus pour compenser la gigue et le changement d'ordre des paquets média.

- Le chapitre 2 présente une revue du protocole SIP en décrivant l'objectif de création du protocole. Pour notre simulation, on étudie un appel basique en analysant les requêtes réponses.
- Le chapitre 3 décrit l'objectif de conception du protocole IAX. On utilise un appel basique pour expliquer en détail les techniques utilisées pour signaler deux parties communicantes puis transférer les données média.
- Le chapitre 4 propose une architecture générale d'environnement expérimental ainsi qu'une analyse des travaux de recherches effectués par d'autres chercheurs dans le domaine.
- Le chapitre 5 décrit la simulation du calcul de la signalisation SIP/IAX et la qualité d'une communication IAX en utilisant les codecs G.711 et GSM. Les résultats obtenus de la performance IAX sont récapitulés à la fin du chapitre.
- Le chapitre 6 décrit la technologie Java 2 Micro Edition (J2ME), utilisée comme moyen d'implémentation du protocole RTP pour permettre un environnement vidéoconférence mobile. La manipulation des paquets RTP sous forme binaire est étudiée par l'outil rtpools [47] et les flux audio et vidéo sont redirigés vers un lecteur multimédia.

Ce mémoire se termine par une conclusion qui résume les résultats du projet et les travaux futurs recommandés.

CHAPITRE 1

LES CARACTÉRISTIQUES DE TRANSMISSION MÉDIA SUR LE RÉSEAU INTERNET

Dans ce chapitre, la première partie fournit un aperçu sur le réseau Internet et ses caractéristiques : l'hétérogénéité et la qualité de service. La deuxième partie se focalise sur le transport et la transmission des médias. Les protocoles TCP ou UDP y sont requis pour n'importe quel transfert. On montre que certaines applications nécessitent TCP alors que d'autres nécessitent UDP et les motifs. On présente également le protocole RTP conçu pour la transmission en temps réel. Enfin, l'architecture de l'implémentation du protocole RTP est illustrée.

1.1 Réseau Internet

Internet est le plus grand réseau mondial. Il connecte des millions d'utilisateurs à travers le monde. Les paquets sont routés de la source à la destination à travers plusieurs sous-réseaux connectés sur un support physique de différentes capacités.

1.1.1 L'hétérogénéité du réseau Internet

L'hétérogénéité d'Internet est due en partie à la diversité architecturale et topologique physique et logique. Les équipements d'interconnexion de différents constructeurs, utilisés à travers tout le réseau mondial Internet affectent d'une façon significative l'homogénéité du matériel. De plus, le lien reliant l'utilisateur final et le fournisseur d'accès a une part d'hétérogénéité, car les liaisons d'accès ont une capacité différente (ADSL, câble, bas débit). C'est cette partie que Floyd et Paxson [31] ont nommée « Last mile problem » car elle est très congestionnée par rapport à d'autres segments dans le cœur du réseau.

La diversité de technologies d'accès n'est responsable que partiellement de l'hétérogénéité, la distance et l'éloignement physique entre les parties communicantes contribue également à

l'hétérogénéité que l'on représente par Round Trip Time (RTT), Phillipa et Sessini [23] ont démontré que la variance (RTT) au niveau du trafic Transmission Control Protocol (TCP) est causée par l'éloignement géographique des utilisateurs, et la variabilité RTT au niveau de la connexion est attribuée au chemin emprunté pour établir la connexion, ainsi qu'à la congestion du segment utilisé sur le réseau.

1.1.2 Les conditions variables du réseau Internet

Le réseau Internet varie selon différentes conditions, principalement dues au trafic généré et au chemin emprunté au cœur du réseau. Les routeurs congestionnés ou plutôt le tampon mémoire débordé résultent en un changement de route. Le changement de route affecte le RTT, le débit, et la perte de paquets. Pour maintenir la stabilité d'un segment utilisé, il faut prévoir une certaine réservation des ressources ou mettre en place la qualité de service.

1.2 Qualité de service

Le protocole IP a été conçu pour transporter les fichiers de données et non la voix ou la vidéo. La seule qualité de service pensée à l'époque est de ne pas perdre ou corrompre les fichiers de données. Par la suite, les progrès technologiques ont permis de transporter la voix et la vidéo en temps réel et il est devenu fondamental de savoir contrôler la qualité de service dans un réseau IP [15]. Les paramètres indispensables qui caractérisent la qualité de service en mode paquet sont :

- La bande passante.
- Le délai de transmission de bout en bout et la variation de ce délai.
- Le taux de perte de paquets et le taux d'ordonnancement de paquets.

Le fournisseur de service doit également s'assurer que la bande passante est répartie équitablement entre les utilisateurs du réseau. Parekh et Gallager [15] ont développé une

approche basée sur les algorithmes de gestion de file d'attente pour assurer un partage équitable de la capacité connue sous le nom Weighted Fair Queuing (WFQ).

La gigue est un élément déterminant pour les applications temps réel. La variation du temps d'arrivée des paquets nécessite un stockage dans un tampon mémoire. Plus l'information arrive de manière non régulière, plus la taille de tampon doit être grande. Par conséquent, un délai supplémentaire est introduit dans le flux d'information de bout en bout.

En général, la perte de paquets est reliée à la bande passante disponible. Un point de congestion produit un débordement de mémoire tampon du routeur qui provoque un rejet et une perte de paquets.

L'ordonnancement de paquets est influencé par l'utilisation de différents chemins pour arriver à la même destination et les algorithmes de gestion de file d'attente lorsque plusieurs interfaces sont disponibles pour une même destination. L'ordonnancement de paquets n'est pas un problème en soi, mais il cause le problème de la gigue.

Les paramètres indispensables de la qualité de service (la bande passante, le délai et la gigue) vont être testés et calculés pour fournir une idée sur la souplesse et la robustesse de protocoles opérants pour la transmission média.

1.3 Transport du média TCP ou UDP et pour quel motif

La couche transport utilise l'un des protocoles (TCP ou UDP) pour fournir des services aux applications. UDP est un protocole simple. Il fournit le multiplexage et le démultiplexage à la couche applicative. Autrement dit, il permet la livraison de données de la source à la destination sans acquittement. Les applications utilisent UDP typiquement pour le streaming multimédia et la téléphonie sur internet.

Le protocole TCP est fiable au contraire d'UDP. Il est capable de retransmettre les segments perdus dans le réseau. Les applications utilisent TCP typiquement pour les courriers électroniques (SMTP), web (HTTP) et le transfert du fichier (FTP).

Le protocole TCP est orienté connexion. Il nécessite l'établissement d'une connexion entre le client et le serveur avant la transmission de données.

Le protocole TCP implémente deux mécanismes pour contrôler le flux transmission [15]: flow control et congestion control.

Flow control permet à la source de limiter le taux d'octets envoyés afin d'empêcher la surcharge du buffer du récepteur.

Congestion control limite le taux d'octets envoyés afin d'empêcher la congestion du réseau et cela est indiqué par le taux de perte de paquets dans le réseau.

1.4 Le protocole temps réel RTP

L'utilisation du protocole temps réel permet de corriger les problèmes dus à la gigue et le changement d'ordre de paquets introduits par le réseau du transport IP. RTP peut être utilisé par n'importe quel type de données média temps réel comme la voix et la vidéo. RTP définit un format spécifique pour les paquets IP transportant les données temps réel.

RTCP (Real Time Control Protocol) est un protocole conjoint du RTP. Il est utilisé pour fournir les informations concernant la qualité réelle de la transmission.

Les protocoles RTP et RTCP n'influencent pas le réseau IP et ils n'ont aucun contrôle sur la qualité de service. Les paquets RTP et RTCP peuvent être détruits ou arriver en désordre. RTP et RTCP permettent à la destination de corriger la gigue et le désordre de paquets via des tampons d'information et les mécanismes de remise en ordre de paquets. Le protocole RTP est utilisé au-dessus du protocole UDP en utilisant un port pair pour le protocole RTP et impair pour le protocole RTCP. Cela n'est pas obligatoire lorsqu'on utilise un protocole de signalisation comme SIP ou H.323.

L'intégration du protocole temps réel RTP dans les applications multimédia nécessite la prise en connaissances de quelques définitions : session RTP, identificateur de source de synchronisation et format NTP (Network Time Protocol) [23] :

- Session RTP est un ensemble de participants communiquant au moyen du protocole RTP.
- Identificateur de source de synchronisation chaque source d'un flux RTP doit disposer d'un identificateur de source de synchronisation SSRC (Synchronization Source) de 32 bits, les paquets de même SSRC ont une référence temporelle et un numéro de séquence commun.
- Format NTP : le marqueur temporel utilise ce format pour indiquer le temps écoulé depuis janvier 1900 à 00H00, il utilise 32 bits pour la partie entière et 32 bits pour la partie fractionnaire.

1.4.1 Format d'un paquet RTP

RTP utilise UDP pour transporter les données média. Les paquets perdus ne sont pas récupérés, car le protocole UDP est non fiable. L'application se charge de gérer les paquets perdus. UDP utilise un numéro du port pour cibler la destination associée avec une adresse IP destinataire (figure 1.1).

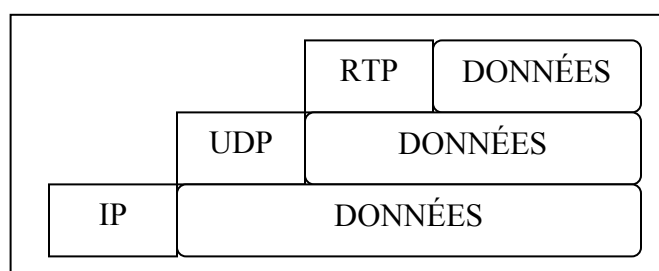


Figure 1.1 Le format d'un paquet RTP.

La taille initiale d'entête du protocole RTP (figure 1.2) est de 12 octets, les champs les plus importants sont: type de contenu, numéro séquence, marqueur temporel et identificateur de source de synchronisation. Les champs identificateur de source contributive (CSRC), profil et la longueur sont extensifs [19].

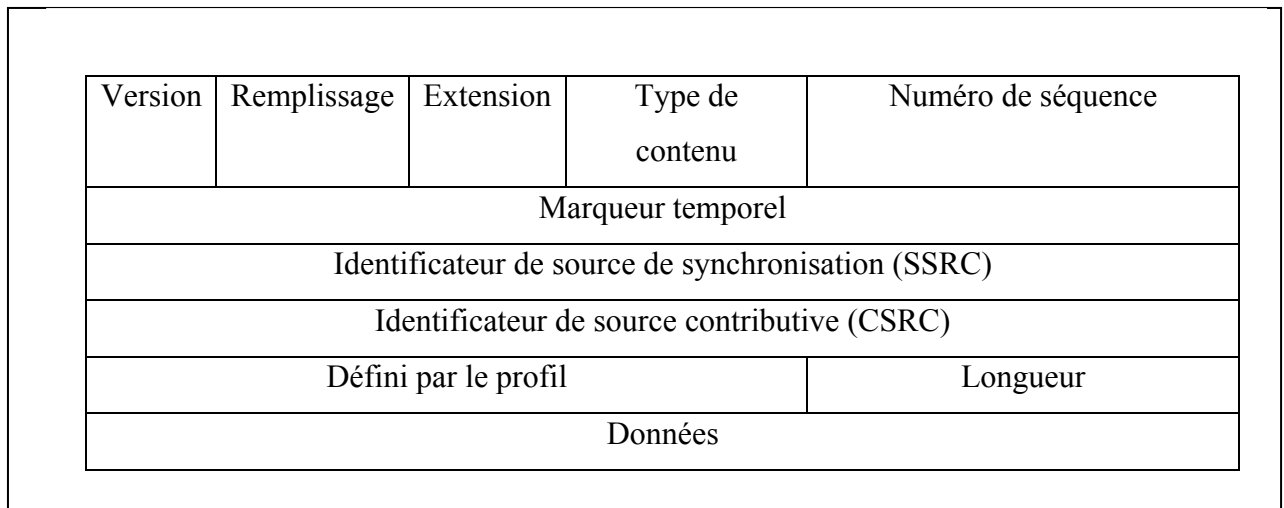


Figure 1.2 L'entête d'un paquet RTP.

Le type de contenu est défini sur 7 bits, il indique le format de l'information transportée dans les paquets sans les analyser. Cela peut être défini par l'application ou un profil RTP.

Le numéro de séquence de 16 bits est initialisé avec l'horloge à des valeurs aléatoires puis incrémenté pour chaque paquet RTP.

Le marqueur temporel de 32 bits utilise une unité définie pour chaque type de contenu.

Identificateur de source de synchronisation (SSRC).

1.4.2 Architecture d'intégration du module RTP

Les applications souhaitant émettre et recevoir les données média, nécessiteront une composante logicielle RTP. En revanche, plusieurs terminaux disposent de périphérique multimédia capable de capturer la voix et la vidéo sans pouvoir la transmettre en temps réel. Nous proposons une architecture générique qui répond aux besoins d'émission et réception des paquets RTP en temps réel tout en prenant en considération les moyens disponibles pour chaque type de terminal.

La composition de paquets RTP nécessite l'accès au tampon contenant les données média à transmettre suivi de la fragmentation de données en fonction du débit de transmission, mais dans la plupart de cas on utilise un échantillonnage de 20 ms pour la voix.

Les paquets fragmentés doivent se synchroniser avec l'horloge d'échantillonnage, et incrémenter après chaque envoi, cette procédure se traduit réellement par l'insertion de l'entête de paquets RTP. (Voir la figure 1.3).

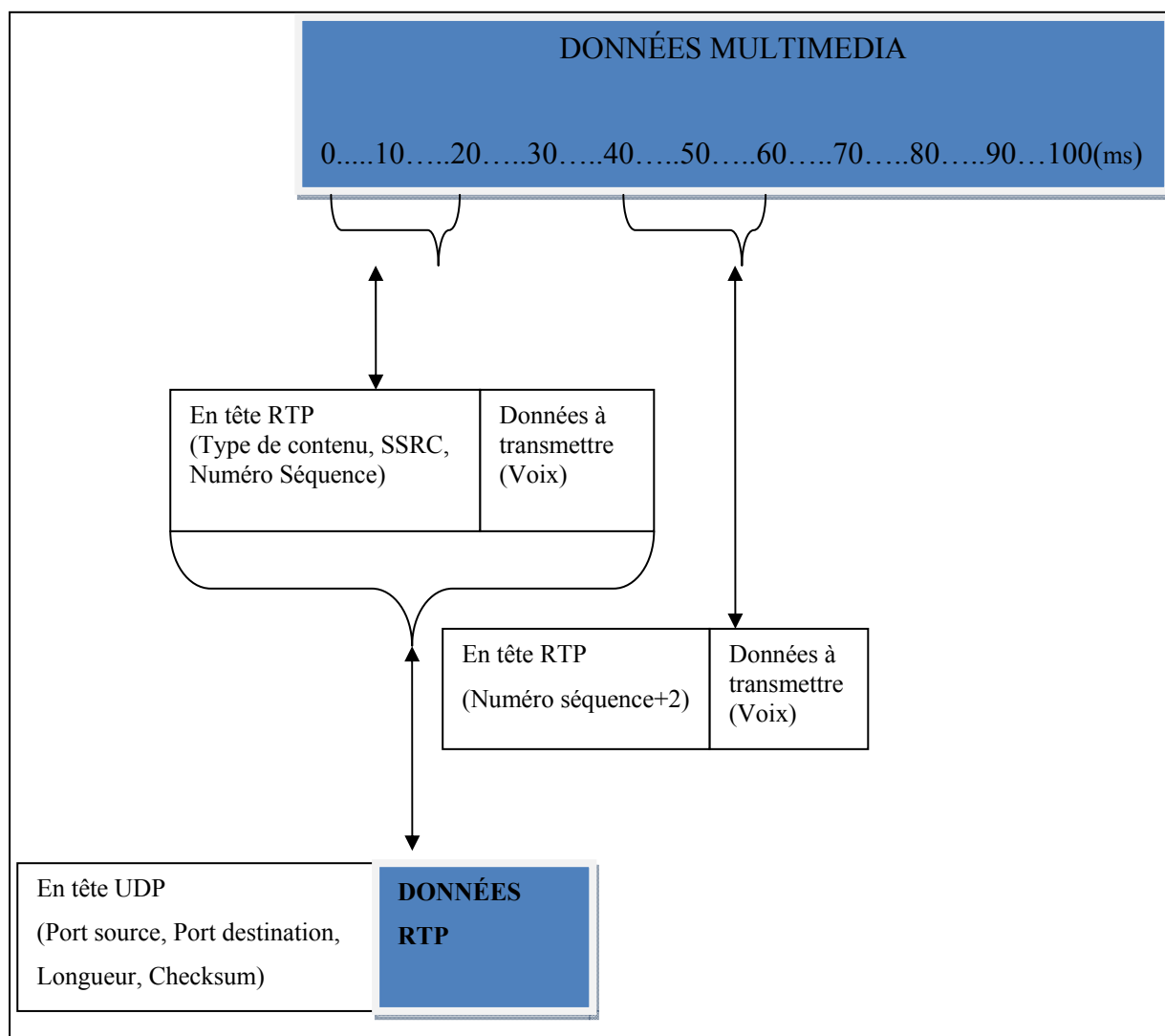


Figure 1.3 La composition d'un paquet RTP.

1.5 Les codecs audio les plus utilisés en transmission média

Les codeurs audio utilisés sont standardisés par (UIT) l'Union International Télécommunication.

Le chapitre suivant présentera plus de détails sur la façon d'encoder et decoder l'audio et la manière de traiter les signaux numériques. Ici on fournit les données importantes de trois différents codecs : G.711, G.726 et G.729.

G.711 quantifie le signal selon deux échelles logarithmiques : la loi A en Europe et la loi μ aux États Unis et au Japon. L'audio codé résulte un débit donné de 64 kbit/s.

GSM (Global System for Mobile communications) représente un codec de parole standard (utilisé à travers le monde) pour la téléphonie cellulaire. Le signal de la voix est échantillonné à toutes les 20 ms pour une fréquence d'échantillonnage de 8000 Hz et un débit du codec de 13 kbps.

La liste de codecs mentionnés (voir le tableau 1.1) est définie dans un profil RTP pour les conférences audio et vidéo. Leurs numéros sont attribués dans RFC 3551 [29].

Tableau 1.1 Le type de contenu pour l'encodage audio

Type de contenu	Codec
0	G.711 loi μ
8	G.711 loi A
3	GSM

1.6 Codecs vidéo

Les codecs vidéo utilisent la représentation des couleurs et le format d'image pour présenter une image. En fait, chaque couleur est un mélange de trois couleurs rouge, bleu et vert (RGB). Le format d'image choisi par la plupart des codeurs pour les applications de vidéoconférence est le CIF (Common Intermediary Format). Il définit une image de 352 par 288 pixels car il fait référence à un format d'écran habituel. D'autres formats sont utilisés pour la transmission à bas débit dont le format QCIF (Quarter Common Intermediary Format) avec une résolution de 176 par 144 pixels, le format QVGA (Quarter of VGA) avec une résolution de 320x240 et le format SQCIF (Sub-Quarter Common Intermediary Format) avec une résolution de 128 par 96 pixels. Les applications professionnelles nécessitent une résolution plus grande alors les images peuvent être codées avec le format VGA (640x480), 4 CIF (704 par 756 pixels) et 16 CIF (1408 par 1152 pixels).

Les codecs vidéo exploitent les redondances spatiales et temporelles des images afin de diminuer leurs tailles. Les codecs vidéos les plus utilisés dans la transmission média en temps réel appartient au standard (UIT) Union International Télécommunication ou aux normes MPEG (Motion Picture Expert Group) d'ISO:

H.263 [15] est une extension de H.261 pour la vidéo à bas débit. Il peut produire des images vidéo à une dimension réduite à un débit allant de 20 à 64 kbps adaptés aux terminaux mobiles.

MPEG-4 peut être utilisé pour une foule d'application allant du codage bas débit à la télévision haute définition.

Le codage H.264 [15] ou MPEG-4 AVC commence à s'imposer en streaming vidéo. Toutefois, sa complexité d'encodage est encore trop grande pour l'utiliser dans les applications de vidéo-conférences mobiles.

Le codec vidéo H.263 va être utilisé pour tester l'intégration du protocole RTP dans les terminaux mobiles.

Le tableau 1.2 montre que chaque codec vidéo correspond à un type de contenu.

Tableau 1.2 Les codecs d'encodage vidéo

Type de contenu	Codec
34	H.263
31	H.261

1.7 Conclusion

Le réseau Internet se caractérise par l'hétérogénéité, car différents types d'accès au réseau sont implémentés. La diversité matérielle et logicielle affecte la qualité de service des applications, qui n'est pas garantie pendant la transmission.

L'architecture physique et logicielle du réseau Internet a permis d'ouvrir de nouvelles pistes de transcodage, d'envisager de nouvelles techniques de compression pour alléger la charge du réseau et rendre la communication audiovisuelle possible.

Ce chapitre présente également l'état de l'art des protocoles de transport TCP et UDP, leur fonctionnement et leur intégration avec d'autres protocoles sur le réseau IP.

Le protocole RTP est conçu pour transmettre en temps réel la voix ou la vidéo encodée, il utilise un profil indiquant le type de contenu et le codec associé. Le protocole RTP permet également la compensation de la gigue et mettre en ordre les paquets reçus.

Le média peut être compressé selon des codeurs audio et vidéo, les plus utilisés pour la voix sont G.711 et G.729. Pour la vidéo en temps réel, H.263 et MPEG-4 sont utilisés.

CHAPITRE 2

SIP, LE PROTOCOLE DE SIGNALISATION MULTIMÉDIA

Ce chapitre présente l'état de l'art du protocole SIP (Session Initiation Protocol) conçu pour initier et terminer une session multimédia. Dans les sections du chapitre, on présentera l'objectif de développement et les révisions qui sont apportées à la version initiale.

Pour notre évaluation du délai signalisation, on étudiera un appel basique en analysant les requêtes réponses. On montrera également le diagramme de séquence pour implémenter le client mobile vidéoconférence.

2.1 L'objectif de développement du protocole SIP

Le protocole d'initiation de session a été défini dans le RFC 2543 [15] par le groupe Multiparty Multimedia Session Control (MMUSIC) appartenant à IETF. L'objectif principal est d'encadrer les protocoles opérant pendant une communication audio ou vidéo : le protocole description de session (SDP), le protocole annonce d'une session (SAP) et le protocole de transmission temps réel (RTP).

Le RFC d'origine a défini SIP pour gérer les sessions multimédias en intégrant la localisation des utilisateurs via leur adresse IP, la disponibilité des usagers, les capacités média supportées par les terminaux et enfin la gestion de la session (initiation, modification de paramètres et finalisation d'une session) [28].

Le protocole SIP est devenu efficace et rapide pour un établissement d'une session, il nécessite un aller et deux retours pour ouvrir un canal média entre des utilisateurs.

Le protocole SIP est capable de transporter les informations médias de la session sans pouvoir les traiter.

2.2 Établissement d'une session

Le rôle principal de l'utilisation du protocole SIP est d'établir une session entre les usagers. Cela nécessite la détermination de l'emplacement des terminaux, la négociation de paramètres médias, la disponibilité de terminaux engagés en cours de communication, et enfin la gestion de communication. L'architecture du protocole SIP dispose de toutes les composantes pour rendre la session efficace.

2.2.1 Architecture SIP

L'architecture du protocole SIP est définie dans RFC 3261[28]. Elle utilise les protocoles Internet Protocol (IP), User Datagram Protocol (UDP) et Transmission Control Protocol (TCP) pour permettre une communication audio ou vidéo. D'autres protocoles peuvent collaborer tels :

- RTP (Real Time Protocol) permet la transmission de données en temps réel pendant une session média.
- SDP (Session Description Protocol) décrit les paramètres d'une session média.
- SAP (Session Announce Protocol) annonce les sessions média en mode multicast.
- RSVP (Ressource Reservation Protocol) réserve les ressources nécessaires.

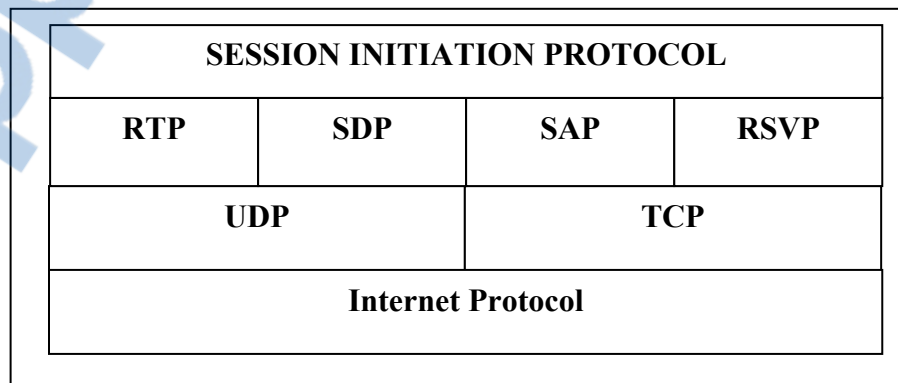


Figure 2.1 L'architecture du protocole SIP.

Le concept client/serveur est appliqué lors de l'implémentation du protocole SIP.

Le client est un composant physique qui peut être un PC, un téléphone mobile, une passerelle ou sous forme générale un terminal dispose d'une pile protocolaire SIP client. Il dispose de deux composantes logicielles :

- UAC (User Agent Client) : envoie les requêtes de traitement à un serveur SIP et reçoit les réponses appropriées.
- UAS (User Agent Server) : reçoit les requêtes d'un client SIP et envoie les réponses correspondantes.
- Il existe quatre types de serveurs SIP capables de traiter les requêtes des clients SIP :
- Registraire : permet à un client SIP de s'enregistrer et être vu pour une éventuelle communication. Il permet aussi l'authentification si nécessaire.
- Proxy : lorsque l'adresse de destination est inconnue, l'expéditeur envoie la requête au proxy auquel est reliée pour réagir et transmettre la requête à la destination appropriée.
- Redirect Server : permet de rediriger une requête destinée au client à plusieurs sorties lorsque ce dernier est mobile. La requête est redirigée au mandataire auquel le client est relié.
- Localisateur : il fournit l'emplacement actuel du client. Ce service est utilisé par le mandataire ou le registraire.

2.2.2 Méthodes

Les méthodes suivantes sont utilisées pour échanger les informations entre les différentes entités (voir RFC 3261 [15]) :

- INVITE : cette méthode permet d'inviter un utilisateur ou un terminal à une session média.
- ACK : permet de confirmer la réception de la réponse finale d'une requête INVITE.
- BYE : cette méthode permet de terminer la session média.
- CANCEL : annulation d'une requête invalide.
- OPTIONS : mentionner la liste de capacités des serveurs.

- REGISTER : enregistrer l'adresse associée au champ « To : » dans le serveur SIP auquel l'expéditeur est relié.

2.2.3 Réponses de requêtes

Les réponses possibles lorsqu'on envoie la requête peuvent appartenir à six catégories de codes. Le bon traitement de la requête est indiqué par le code 2xx.

La liste de codes possibles est :

- 1xx : indique que la requête est reçue et en cours de traitement.
-
- 2xx : indique que le traitement de la requête a bien été effectué.
- 3xx : Informations supplémentaires pour traiter la requête.
- 4xx : indique que le client a envoyé une requête avec une erreur de syntaxe.
- 5xx : indique que le serveur ne peut pas traiter la requête
- 6xx : indique que la requête ne peut pas être traitée.

2.2.4 Format de messages

Les messages SIP sont basés sur le format texte, ils utilisent l'encodage UTF-8 en mode caractère défini dans (RFC 2279) [34]. Un message SIP peut être une requête d'un client attaché à une session ou une réponse d'un serveur. Les messages de requête et réponse utilisent le format standardisé dans le RFC 2822[34]. Les messages SIP consistent en :

- Début de ligne.
- Un ou plusieurs champs d'en-tête.
- Un saut de ligne pour indiquer la fin des champs d'en-tête.
- Le corps message est optionnel.

SIP/2.0 200 OK

Contact :

To :

From :

Allow :

Content-type :

Content-Length

v= 0

o=-4 2 IN IP4 192.168.1.5

Cette exemple montre un message SIP sous forme une réponse de la requête, les champs d'entête : contact, To, From, Allow, Content-Type et Content-Length. Le corps du message est v=0 et o=-4 2 IN IP4 192.168.1.5.

2.2.5 L'enregistrement du client SIP

La partie expérimentale du projet inclura à mesurer le temps requis pour l'enregistrement en se basant sur les statistiques expérimentales. Le diagramme de séquences utilisé pour calculer le temps d'enregistrement est le suivant (figure 2.2) :

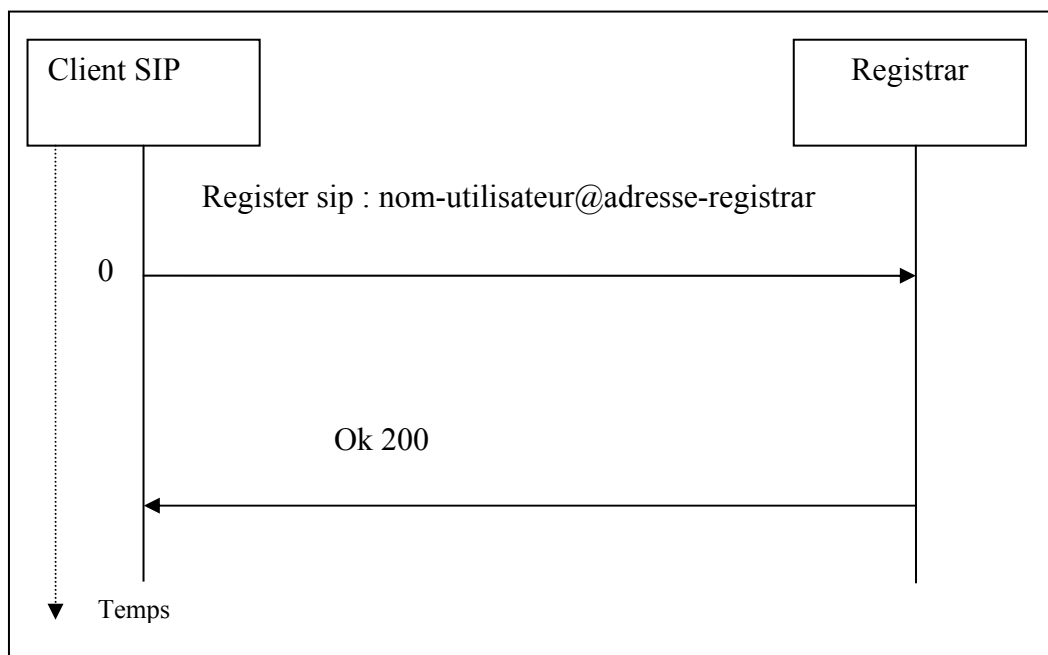


Figure 2.2 Les séquences d'enregistrement (le délai Setup).

Le client SIP envoie un message Register et reçoit un acquittement Ok. Le délai d'enregistrement est la durée entre l'envoi du message Register et la réception du message Ok.

2.2.6 Les messages de la signalisation

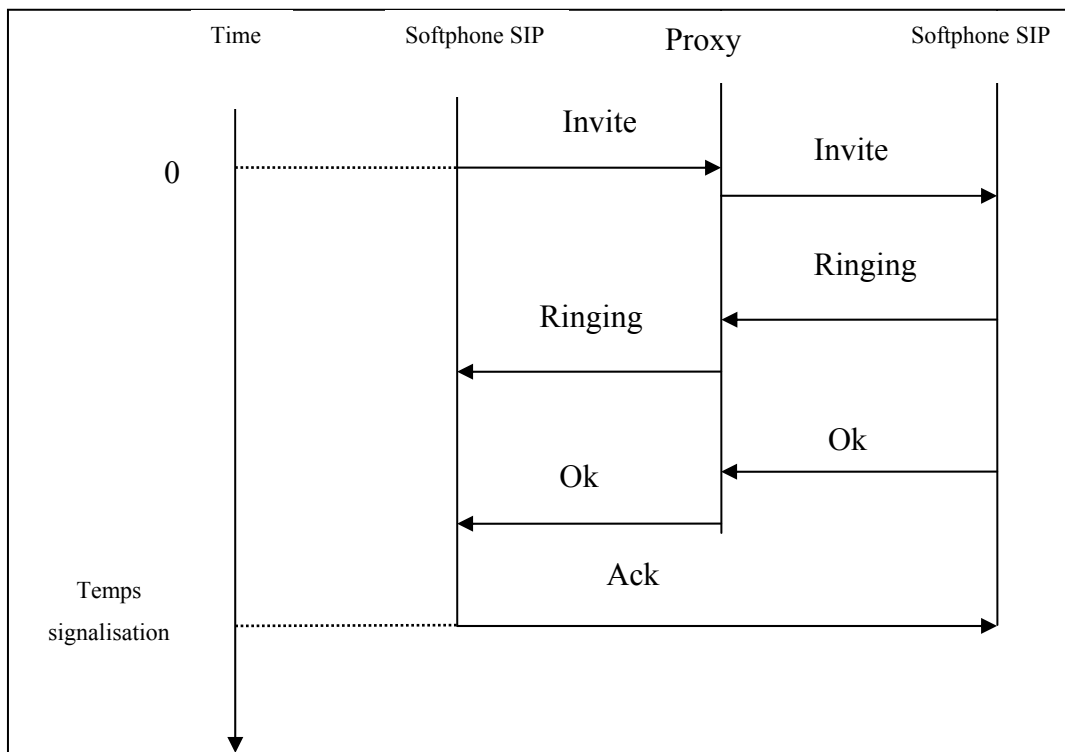


Figure 2.3 Le diagramme de signalisation SIP (le délai Setup).

La signalisation d'un client SIP se compose d'un aller et deux retours de requêtes (invite, ringing et ok) (figure 2.3). Le temps mesuré pendant la procédure de la signalisation, commence du moment d'envoi de la méthode INVITE et finit à la réception du message Ok du client 2. À ce moment, la négociation de paramètres est effectuée entre les deux clients et on considère que la signalisation est terminée.

2.3 Conclusion

Dans ce chapitre, nous avons vu le protocole SIP destiné à la signalisation des usagers dans une session multimédia. Nous avons montré les diagrammes de séquence utilisés pendant ce projet pour mesurer le temps requis d'enregistrement et de signalisation.

CHAPITRE 3

IAX, LE PROTOCOLE DE TRANSMISSION MÉDIA SUR LA PLATEFORME IPPBX

Ce chapitre présente le protocole de transmission média IAX, conçu pour l'utilisation entre IPPBX. On explique son fonctionnement en détaillant un exemple d'appel basique. Ensuite, on présente les éléments que nous utiliserons pour calculer le temps requis d'enregistrement et de signalisation.

La description du protocole dans ce chapitre est basée sur le draft IAX version 5 publiée électroniquement sur le site : www.ietf.org car le protocole n'est pas standardisé.

3.1 L'objectif de conception

Le protocole IAX (Inter-Asterisk Exchange) est conçu pour fournir le contrôle et la transmission de données média sur Internet. IAX a le rôle d'un protocole contrôleur et transporteur de données média. Il est destiné spécialement pour les appels voix sur IP.

Les objectifs principaux décrits dans le (draft-iax-05) sont :

- Minimiser l'utilisation de la bande passante qu'elle soit pour contrôler ou transmettre les données média.
- Fournir une transparence de translation d'adresse réseau (NAT).
- Transmettre les informations de numérotation (Dialplan)
- Implémenter les caractéristiques d'Interphone et paging.

IAX est considéré comme protocole « all in one » pour manipuler les données multimédias. Il combine le contrôle, la transmission de données média dans un seul protocole. De plus, IAX utilise un port unique et statique pour simplifier la traversée et la translation d'adresse IP, et aussi pour éviter l'utilisation d'autres protocoles autour du serveur NAT.

Le protocole IAX utilise un entête compact pour minimiser l'utilisation de la bande passante. Sa nature open source permet également l'ajout de nouveaux types de contenu pour supporter des services supplémentaires.

3.2 Appel basique IAX

Le fonctionnement d'un appel basique entre deux entités IAX va nous permettre de comprendre les procédures d'enregistrement, de signalisation et de transmission média. L'utilisation de la plateforme IPPBX avec deux clients IAX nous aidera à calculer le temps requis pour l'enregistrement et la signalisation.

Ensuite, les paramètres de la transmission média vont être testés et calculés pour fournir un aperçu de la qualité de service de communication IAX.

3.2.1 La séquence de messages d'enregistrement

L'enregistrement permet à un client IAX de s'enregistrer et de fournir son adresse IP et ses informations d'authentification à l'appelé.

Le protocole IAX permet l'authentification via différentes méthodes :

- MD 5 (Message Digest Authentication) (RFC 1321) [34] utilise la somme d'arrangement md5.
- RSA (Rivest, Shamin Adleman) est un algorithme qui utilise la clé publique/privée (RFC 3477) [34].

L'enregistrement consiste à envoyer les informations d'authentification au serveur d'enregistrement (Registrar) via un message (REGREQ). Si les informations d'authentications sont valides, le serveur envoie un message (REGACK) en demandant d'indiquer l'adresse qui sera utilisée.

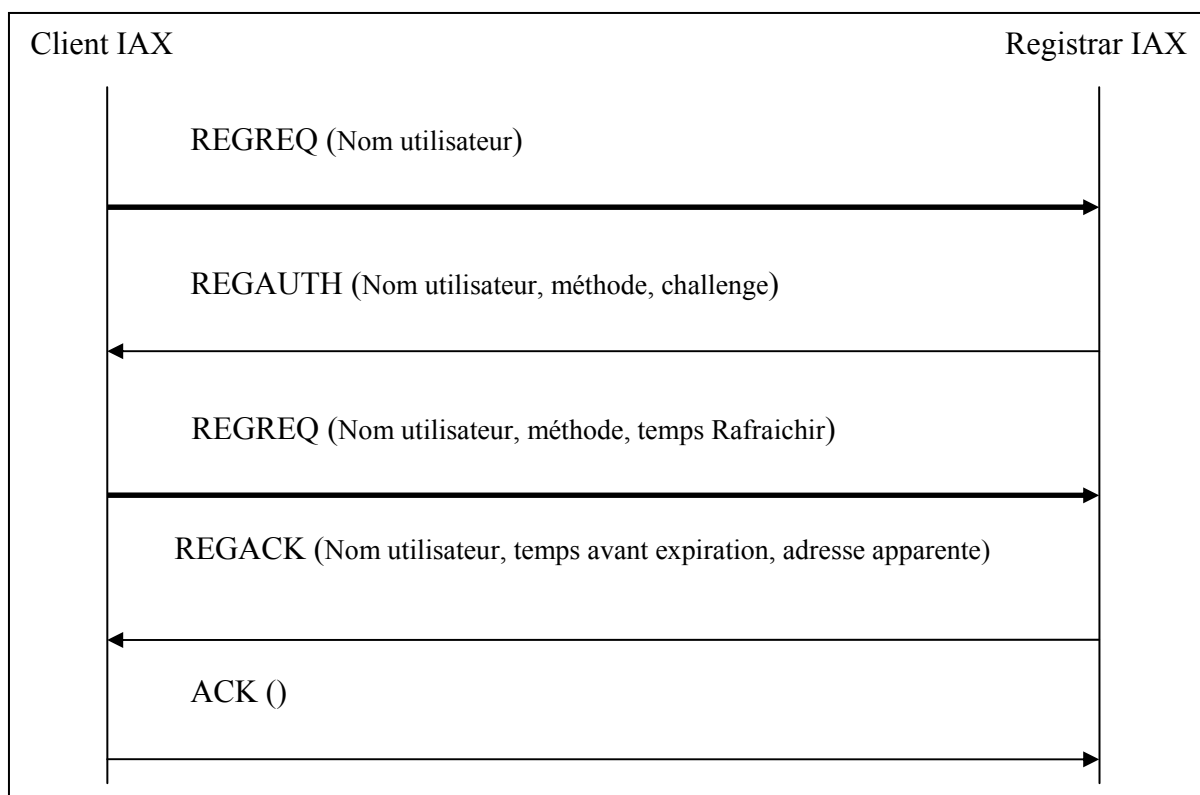


Figure 3.1 Les requêtes d'enregistrement IAX.

Tirée du draft-05 publié électroniquement [34].

Les requêtes utilisées pendant l'enregistrement d'un client IAX sont :

REGREQ : c'est une requête d'enregistrement indépendante du média supporté. Elle est utilisée pour la demande d'enregistrement et pour la réponse à une requête d'authentification. Le message doit contenir le nom d'utilisateur, et optionnellement le temps nécessaire pour rafraîchir l'enregistrement. La méthode utilisée pour l'authentification est déterminée à partir du serveur d'enregistrement. Cette méthode est également incluse.

REGAUTH : authentification d'enregistrement est une réponse aux requêtes d'enregistrement et libération d'enregistrement. Ce message permet l'authentification d'une entité demandant l'enregistrement ou la libération d'enregistrement. Il doit contenir le nom d'utilisateur, la méthode d'authentification (MD5 ou RSA), et la clé associée.

REGACK : le message accusé de réception d'enregistrement est envoyé, lorsque la procédure s'est bien déroulée. Il peut contenir le temps en secondes avant l'expiration d'enregistrement.

Le temps requis de l'enregistrement d'une entité IAX est calculé en se basant sur les messages REGREQ et REGACK. Le diagramme de séquence d'enregistrement du client IAX est dans la figure 3.2.

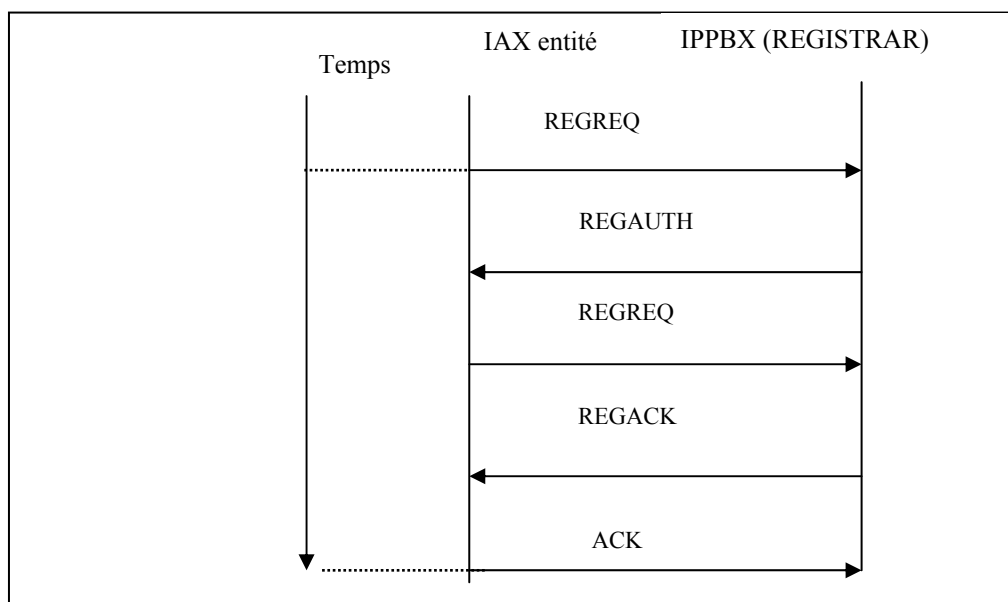


Figure 3.2 Les séquences d'enregistrement IAX pour calculer le délai Setup.

3.2.2 La signalisation de terminaux IAX

Le protocole IAX peut être utilisé pour mettre en place une liaison entre deux clients IAX avant d'établir la communication. Cette procédure appelée « Call legs » se compose de deux messages principaux :

- **Message New** : contient les informations de destination. La réponse peut être une demande d'authentification, rejet de demande ou « Call legs » est établi.

- Message **Accept** : indique que l'établissement d'appel au niveau « Call legs » s'est bien déroulé.

Call legs permet de relier les deux entités communicantes et faire passer les messages de contrôle d'appel jusqu'à la fin d'appel (figure 3.3).

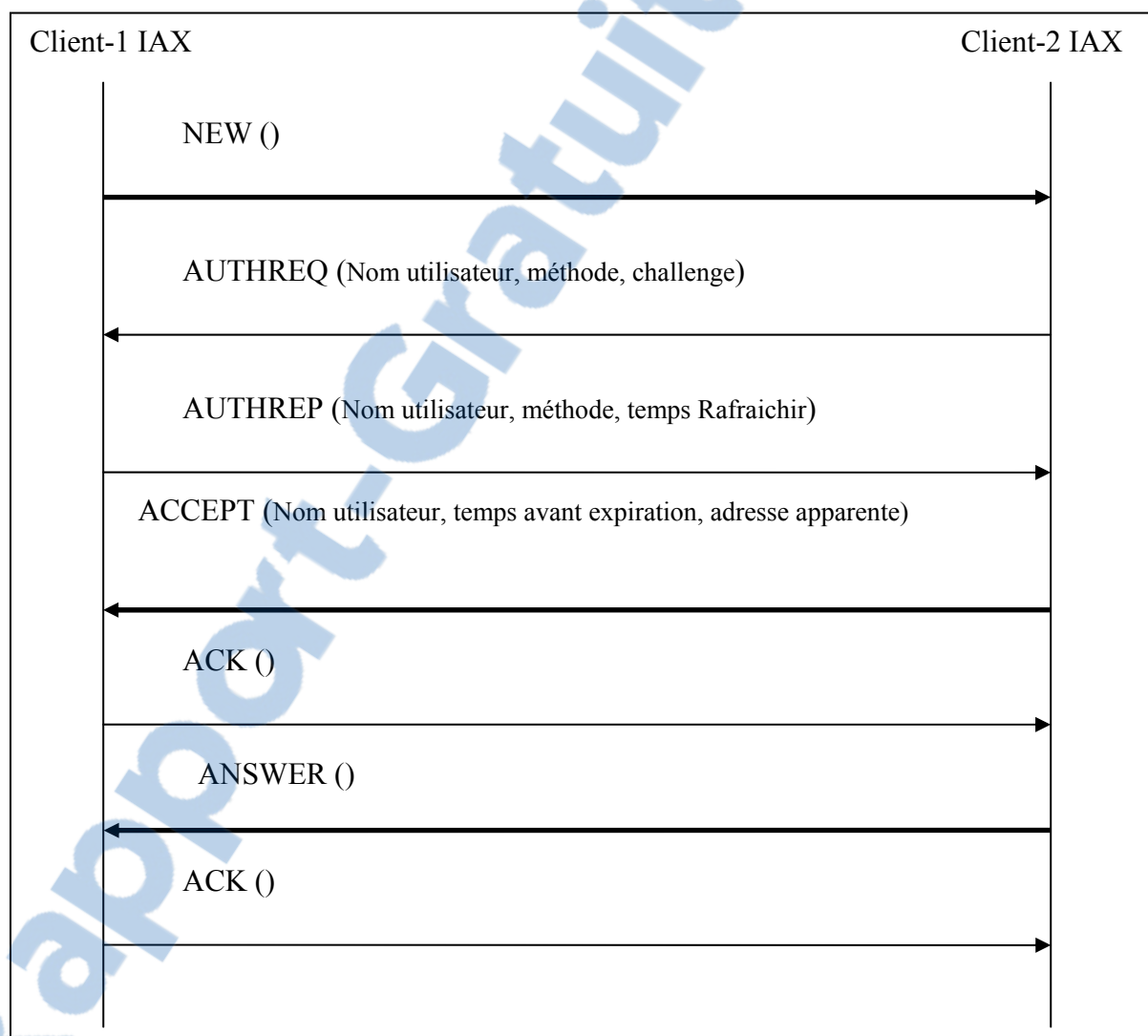


Figure 3.3 Les séquences de signalisation IAX.
Tirée du draft-05 publié électroniquement [34].

Remarque : Nous avons utilisé la durée écoulée entre les messages NEW et ACCEPT pour calculer la durée de signalisation IAX.

L'établissement de l'appel via le protocole IAX (Call Leg) nécessite deux requêtes (NEW et ACCEPT). Les requêtes AUTHREQ et AUTHREP ne sont pas nécessaires à moins que le client IAX ne soit pas authentifié au début de la signalisation.

On considère dans l'expérience de signalisation que le client est déjà authentifié, donc la signalisation nécessite deux requêtes NEW et ACCEPT pour établir la signalisation. Le message ACCEPT inclut l'un des codecs spécifiés par la requête NEW.

3.2.3 Mini-Frame

Les paquets transportant la signalisation IAX et les données média utilisent un port unique 4569 au dessus du protocole de transport UDP. Le draft-iax définit deux types de paquets :

- Full Frame : peut être utilisé pour transporter les données de signalisation et les données média. La structure Full Frame est optimale pour initialiser, contrôler et terminer un appel IAX car les paquets Full Frame sont fiables. Ils nécessitent un accusé de réception immédiat après la réception. La longueur de l'entête qui compose Full Frame est de 12 octets.
- Mini Frame : est utilisé pour transporter les données média uniquement. La taille de l'entête Mini Frame est de 4 octets. Les paquets Mini Frame sont non fiables.

Nous avons enregistré les paquets Mini Frame capturés dans un fichier pour être analysés ultérieurement durant le calcul de la performance IAX. Nous avons testé les paramètres de qualité de service : la bande passante, le délai et la gigue.

3.3 Conclusion

Dans ce chapitre, nous avons vu le protocole IAX destiné spécialement pour la signalisation et la transmission du média sur IP.

Nous avons montré la procédure d'enregistrement et d'établissement d'appel que nous utiliserons pour calculer la durée d'enregistrement et de signalisation IAX.

Nous avons cité les différents types de paquets que nous analyserons pour tester la performance d'une communication IAX.

CHAPITRE 4

ENVIRONNEMENT EXPÉRIMENTAL ET RECHERCHES ANTÉRIEURES

Dans les chapitres précédents, nous avons vu les différents protocoles utilisés pour transmettre les données média.

Ce chapitre propose une architecture générale d'environnement expérimental ainsi qu'une analyse des travaux de recherches antérieures dans le domaine.

La plateforme IPPBX est un outil assez complet pour tester la transmission média sur les réseaux IP en utilisant les protocoles SIP sur RTP et IAX. Elle sera utilisée pour nos travaux.

4.1 Les exigences de l'environnement expérimental

L'environnement expérimental de la transmission média sur les réseaux IP en utilisant les protocoles SIP et IAX doit contenir le client, le serveur proxy et le capteur de trames.

Le client peut être un logiciel qui utilise le protocole SIP ou IAX pour signaler avec d'autres clients, et transporte les données média par l'intermédiaire du protocole RTP ou IAX mini frames. On peut distinguer deux types du client :

- Le client matériel : il peut être un téléphone IP ou un téléphone analogique connecté à un adaptateur ATA.
- Le client logiciel : il est sous forme d'un progiciel à installer sur l'ordinateur.
- L'analyseur de trames : il permet de capturer les paquets circulant entre les clients, il est capable de fournir les informations identifiant les données média et aussi les performances d'une communication SIP ou IAX.

Pour notre projet nous avons quelques contraintes, que l'on peut définir comme suit :

- Le serveur proxy doit être capable de faire communiquer les clients SIP et IAX, cela est nécessaire pour pouvoir calculer le temps requis de signalisation SIP et IAX. Le serveur doit commuter également les différents canaux SIP et IAX.
- Le capteur de trames doit être capable d'identifier les informations du protocole SIP et IAX, et aussi il doit fournir un environnement de développement de nouveaux scripts
- Les outils disponibles nous limitent à utiliser le client logiciel.
- L'émulateur mobile doit avoir les progiciels nécessaires pour la signalisation SIP, l'accès au réseau et le traitement médias.

4.2 Les éléments de l'architecture testés et les alternatives

Dans un premier temps, nous avons testé SIP Express Router [54], qui fait le rôle d'un proxy, registraire et Redirect SIP. Sa mise en œuvre est conviviale. Il offre toutes les fonctionnalités pour tester la transmission média sur les réseaux IP en utilisant le protocole SIP. Il peut être combiné avec RTP proxy [55] pour offrir un relais média.

Ensuite, nous avons testé IPPBX Asterisk [35], qui fait le rôle d'un autocommutateur complet. Il supporte les protocoles SIP et IAX pour transporter la voix ou la vidéo sur les réseaux IP. Il est interopérable avec le système téléphonique traditionnel (PSTN) et le réseau cellulaire GSM pour de futurs travaux.

L'émulateur mobile doit être capable de simuler un terminal mobile (téléphone mobile, Smart phone, et PDA) doté des caractéristiques spécifiques en termes des ressources mémoires et puissance de calcul. Nous avons testé plusieurs solutions : Nokia, Motorola, Windows Mobile et Sun Wireless Toolkit. Il existe une multitude de différences au niveau du système d'exploitation et du langage de programmation. Symbian est un système d'exploitation intégré dans la plupart de terminaux mobiles de la troisième génération. Il offre des fonctionnalités avancées pour tester la transmission média sur les réseaux IP au niveau de l'encodage et du décodage média. Le protocole de signalisation SIP et le protocole

de transport média RTP sont intégrés au terminal mobile sous forme de solution propriétaire et adaptable au lecteur multimédia.

D'autre part, la technologie J2ME utilise le concept de la machine virtuelle JAVA dédiée aux terminaux mobiles. Elle est disponible sur la plupart de terminaux, cette technologie utilise JSR (Java Specification Request) pour couvrir un ensemble de fonctionnalités. JSR 180 apporte les éléments nécessaires pour signaler un terminal mobile dans une session multimédia via le protocole SIP. JSR 118 offre les fonctions nécessaires pour accéder au réseau IP et manipuler les datagrammes UDP. JSR 135 gère la capture, le traitement et l'affichage de la voix et de la vidéo.

Les clients logiciels testés sont : Xlite [48], iaxlite [50], ekiga [51], samplephone [52] et Zoiper-Communicator [53]. Les fonctionnalités supportées par les clients logiciels se ressemblent : enregistrement au serveur, rejoindre une session multimédia, transport de la voix ou de la vidéo, les codecs audio et vidéo supportés sont limités aux codecs non payants. Le client logiciel Xlite combine le transport de la voix et la vidéo, il est capable d'établir une communication audio ou vidéo à n'importe quel client matériel (téléphone ou téléphone IP) ou logiciel.

L'analyseur de trames est un outil indispensable pour visualiser et contrôler le contenu des données transmises. Nous avons testé Ethereal [49], WireShark [38], et Unsniff network. Ethereal [49] peut être utilisé pour analyser les trames venant de différents réseaux : ATM, FDDI, Ethernet et réseau sans fil. Il est supporté sur différents mainframes mais la solution Ethereal est plus adaptée au système d'exploitation Linux. WireShark [38] analyse la communication SIP et nous pourrions le configurer pour qu'il puisse calculer le temps requis pour l'enregistrement et la signalisation. Les performances d'une communication SIP peuvent être calculées en utilisant les fonctionnalités appropriées. Le flux média peut être capturé et enregistré sous plusieurs formats disponibles avec l'outil WireShark. Unsniff Network Analyser [43] est capable d'analyser les trames. Il offre la possibilité de développer

des scripts spécifiques aux nouveaux protocoles. Cet outil est capable d'analyser le flux média en temps réel.

4.3 Contexte de l'environnement expérimental et

L'architecture générale du projet contient deux axes de travail :

Axe 1 : l'installation de la plateforme IPPBX, le calcul du temps requis de la signalisation SIP/IAX et la performance d'une communication IAX en utilisant les codecs G.711 et GSM.

Axe 2 : L'implémentation d'un client mobile pour la vidéoconférence (plus de détails au chapitre suivant).

L'architecture de composants techniques est dans la figure 4.1.

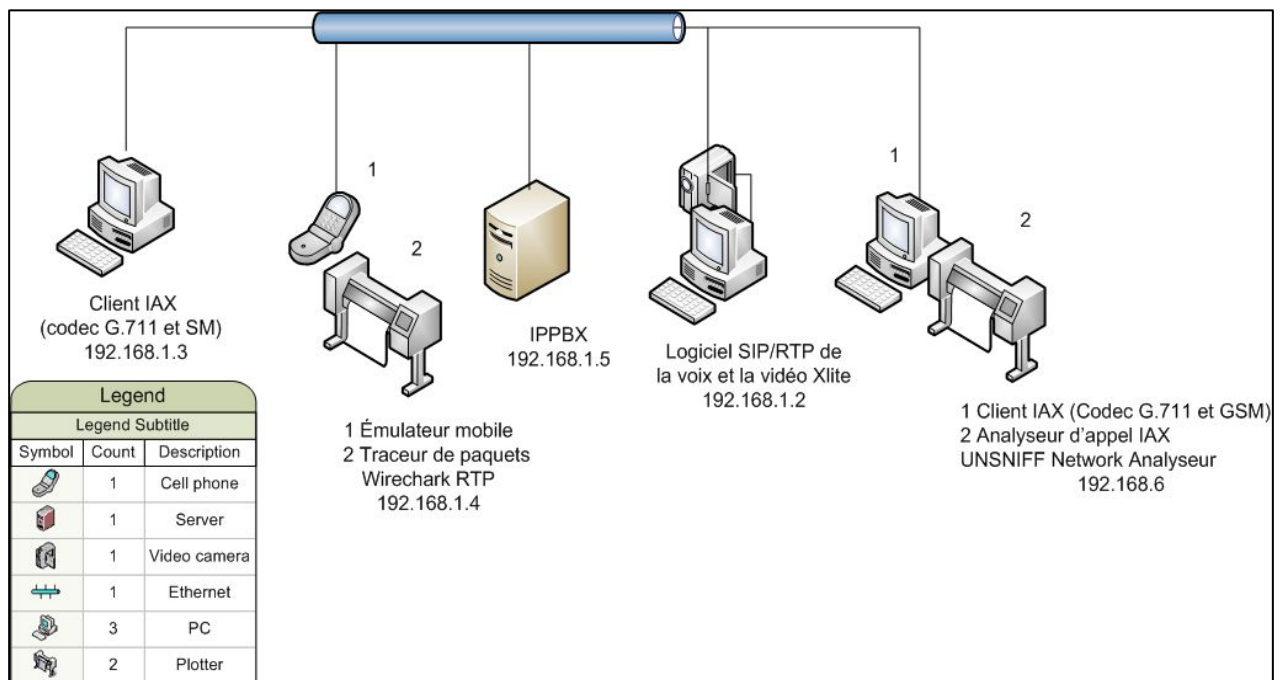


Figure 4.1 L'architecture de composants techniques.

La figure 4.1 montre l'architecture de l'environnement expérimentale, qui se compose en :

- Traceur de paquets WireShark [38] permet de capter les trames circulantes entre l'émulateur mobile et le softphone (SIP/RTP). Cet outil va nous permettre d'analyser le média capté pendant une session multimédia. Il va aussi nous servir pour calculer le temps requis pour l'enregistrement et la signalisation SIP/IAX. Les fonctionnalités offertes par [38] par rapport [49] sont la possibilité de capturer les requêtes de signalisation SIP/IAX en temps réel, la sauvegarde dans un fichier nous facilite l'analyse et le calcul de délai d'enregistrement et de la signalisation. L'implémentation du protocole RTP dans un terminal mobile nécessite la sauvegarde de flux RTP sous le format *rtpdump* pour sa lecture ultérieurement. L'analyse du flux RTP via l'outil WireShark permettra le calcul de la bande passante le délai et la gigue selon RFC 3550.
- Les scripts de l'annexe I, II et III sont développés avec le langage Ruby et nécessiteront l'environnement de déploiement IAX Unsniff Network Analyseur [41] car l'architecture du capteur de trames est Plug In Play et les scripts développés sont facile à être intégrés. L'interface graphique de l'outil est indépendante du protocole analysé, elle est possible d'être adhérente aux scripts d'analyse du protocole IAX. Nous croyons qu'IAX Unsniff Network Analyseur [43] est le seul outil capable de calculer les performances du protocole IAX.
- L'émulateur mobile utilise la technologie J2ME [37] pour fournir un environnement de test convivial de terminaux mobiles. Ce choix de la technologie est gouverné par la disponibilité de la machine virtuelle Java dédiée aux terminaux mobiles et le développement avancé des APIs nécessaires pour connecter un client mobile de vidéoconférence à une session multimédia dont JSR 180, JSR 118 et JSR 135.
- Le client logiciel de la voix et la vidéo IP utilisé pendant la communication est le softphone (SIP/RTP) Xlite [48], il utilise la caméra pour transporter la vidéo, il peut établir des appels SIP voix ou vidéo. Il supporte les codecs G.711 et H.323 qu'on utilisera pour implémenter un client mobile de vidéoconférence. Les autres clients logiciels tels que ekiga et Zoiper-Communicator ne supportent pas la fonctionnalité de la vidéo.
- Le serveur registraire proxy SIP/IAX est la plateforme Asterisk IPPBX [35], on l'utilisera comme plateforme de test, car elle supporte la commutation de communication SIP/IAX. Au contraire, Sip Express Router [54] ne supporte que le protocole SIP. Les alternatives

pour tester le protocole IAX ne sont pas nombreuses. Il existe des solutions qui combinent le client et le serveur au même progiciel comme Yate [56] mais ils sont tous dérivés de la plateforme Asterisk IPPBX. Nous utiliserons la plateforme Asterisk IPPBX comme serveur proxy SIP/IAX car le protocole IAX a été développé spécifiquement pour résoudre quelques problèmes rencontrés lors du déploiement de la plateforme et pour connecter deux autocommutateurs Asterisk. Nous croyons que cette plateforme pourra être utilisée pour d'autres travaux de recherche et plus spécifiquement pour sa capacité d'être interconnecté au réseau téléphonique traditionnel PSTN et au réseau cellulaire GSM. La plateforme IPPBX est décrite dans la prochaine section avec les travaux antérieurs.

4.4 Description de la plateforme IPPBX et les recherches associées

La plateforme IPPBX permet la commutation de canaux venant d'un réseau IP et la commutation de circuits téléphoniques. Elle remplit le rôle d'une passerelle entre différentes technologies : le réseau téléphonique traditionnel, le réseau Internet et le réseau local.

La connexion de circuits téléphoniques à la plateforme IPPBX se fait par l'intermédiaire d'une carte FXO (Foreign eXchange Office) et FXS (Foreign eXchange Subscriber).

Les protocoles utilisés pour transmettre les médias au niveau de la couche applicative sont : SIP, SDP, RTP et IAX.

Dans l'article [1], on affirme que la plateforme Asterisk est un outil assez complet pour conduire les expériences et les simulations VoIP. Les chercheurs ont effectué une étude comparative et expérimentale du trafic circulant dans la métropolitaine Ottawa. La méthode de mesure utilisée est Mean Opinion Score (MOS). Cette méthode permet la notation des appels en fonction de la qualité (voir le tableau 4.1).

Tableau 4.1 La qualité de l'appel en utilisant MOS (Mean Opinion Score)

La valeur (MOS)	Qualité
1	Faible
3	Moyenne
5	Excellente

La performance de SIP/IAX est testée en fonction de deux paramètres (la perte de paquets et le délai). Onze échantillons d'appels sont utilisés pour obtenir une note d'opinion moyenne (MOS). Les appels sont effectués sous les conditions similaires du réseau.

Les résultats de test d'expérience indiquent que l'appel IAX maintient une qualité supérieure lorsque le taux de perte de paquets varie entre 0,5 et 9%. Sur cet intervalle, les mesures prises montrent que l'appel IAX affiche une amélioration de performances de 0.513 MOS et une amélioration de performance relative de SIP à 24.70 % (calculé selon la formule 4.1).

$$\sqrt{\frac{1}{N} \sum_{i=1}^N \left(\frac{IAX_i - SIP_i}{SIP_i} \right)^2} \quad (4.1)$$

Dans la formule (4.1), IAX_i représente la qualité de la mesure i pour le protocole IAX, SIP_i représente la qualité de la mesure i pour le protocole SIP, et N représente le nombre de mesures prises.

Similairement, la qualité d'appel IAX est améliorée de 7.16 % par rapport à SIP en changeant le délai d'arrivée des paquets dans un intervalle [0, 2000 ms] par pas de 200 ms. La formule 4.1 est utilisée également pour calculer cette amélioration.

D'autres travaux ont été effectués sur le serveur IPPBX Open source pour tester la robustesse et les options de fonctionnement [2] et [3].

L'article [2] examine les étapes d'installation de la plateforme ASTERISK en mentionnant la configuration appropriée pour être intégrée dans un réseau hétérogène (SIP et H.323).

Dans [2], on décrit également l'utilisation de l'IPPBX pour acheminer les appels entrants au réseau local, et aussi rediriger les appels sortants vers le réseau téléphonique traditionnel. L'implémentation [2] comprend *VoIP Gatekeeper* pour fournir la translation d'adresses et le contrôle d'accès à l'intérieur du réseau local. Dans ce projet on utilisera la même procédure d'installation de la plateforme IPPBX, et on ajoutera un script automatisant la compilation et l'installation de la plateforme (Annexe I).

Cependant, l'auteur de l'article [3] explore l'interopérabilité entre Cisco Call Manager et IPPBX. Le résultat montre qu'IPPBX s'intègre parfaitement dans un réseau composé de d'autres types d'autocommutateurs comme Cisco Call Manager.

Les options supportées par un autocommutateur standard : Afficheur, transfert d'appel et répondeur sont testés et fonctionnent correctement [3].

L'aspect sécuritaire de la transmission média est étudié dans [30], l'auteur a décrit les méthodes actuelles adaptées (confidentialité, intégrité et disponibilité) à sécuriser le flux média ensuite il a évalué la performance bande passant et CPU après avoir implémenté chaque méthode de sécurité.

Le point fort IPPBX Open source est la capacité d'intégrer différents modules pour tester la performance réseau [1] et [3], la sécurité [30], le contrôle et la signalisation de services multimédia IP [5].

Notre objectif, dans ce projet, est d'implémenter la plateforme basée sur IPPBX pour pouvoir transmettre de la voix sur IP et de mesurer le temps requis pour l'enregistrement et la signalisation des protocoles SIP et IAX.

Les performances de la communication IAX (la bande passante, le délai et la gigue) sont calculées en exécutant un script d'analyse IAX.

4.5 Installation de l'IPPBX Asterisk

L'installation consiste à mettre en œuvre une machine dotée des progiciels nécessaires du système Asterisk et les logiciels capables de traiter la voix et la vidéo sur IP.

L'installation nécessite la récupération et la compilation des fichiers sources.

Lorsqu'on souhaite utiliser une liaison au réseau téléphonique, d'autres cartes permettront l'interface au réseau PSTN sont nécessaires. À ce moment, la compilation du module Zaptel est fondamentale, car il permet de faire fonctionner les cartes PRI, BRI et FXO/FXS.

4.5.1 Matériels utilisés

Pour l'installation d'Asterisk, nous avons utilisé une machine avec les caractéristiques suivantes :

- Processeur P4 1.8 GHz
- 512 Mo de RAM
- 40 Go de disque dur
- Carte réseau 10/100 Mbit.

Le choix du serveur dépendra en grande partie du nombre de canaux de voix à gérer. Un très grand nombre de lignes demande un serveur relativement puissant.

Les soft phones permettent de jouer le même rôle que les téléphones IP, mais l'utilisateur doit avoir un casque et un micro pour pouvoir passer et recevoir des appels. Il existe plusieurs soft phones gratuits et qui supportent les protocoles SIP et IAX.

4.5.2 Script du déploiement

Asterisk est offert sous forme de plusieurs package adaptés à Linux CentOS version 4.4. Nous avons créé un script d'installation d'Asterisk pour automatiser le déploiement (voir Annexe I).

Nous avons récupéré les bibliothèques nécessaires à l'installation (zlib 1g, zlib 1g-dev, libncurses5, libncurses-dev, libssl0.9.6, libssl-dev, libnewt-dev et libnewt0.51)

La commande d'installation :

```
Apt-get install zlib 1g zlib 1g-dev libncurses5 libncurses-dev libssl0.9.6 libssl-dev libnewt0.51.
```

Ensuite, nous avons récupéré à partir du site de Digium, les archives de Zaptel et Asterisk. Il faut noter que l'on a utilisé la version d'Asterisk 1.4.4 et Zaptel 1.4.2.1 :

```
wget http:// ftp.digium.com/pub/asterisk/releases/asterisk-1.4.4.tar.gz &&
```

```
wget http://ftp.digium.com/pub/zaptel/ tel.tar.gz &&
```

```
wget http://ftp.digium.com/pub/libpri sionlibpri &&
```

```
wget http:// ftp.digium.com/pub/asterisk ons.tar.gz
```

4.5.3 Création d'extensions SIP

Les clients SIP se connectent à IPPBX via les extensions du protocole SIP ensuite ils peuvent établir une session multimédia pour faire passer de la voix ou la vidéo sur IP.

Le fichier *sip.conf* permet d'ajouter les clients SIP. La procédure consiste à remplir les champs prédéfinis tels que :

- *Username* le nom d'utilisateur
- *Secret* le mot de passe
- *Context* le contexte dans lequel le compte est associé dans le fichier extension
- *Type* il existe trois types de compte : *peer* utilisé pour appelé le compte de l'opérateur, *friend* pour envoyer et recevoir des appels, *user* pour recevoir les appels.

- *Host* l'adresse IP du serveur IPPBX
- *Allow* la liste de codecs autorisés pour les paquets audio

Un exemple de configuration d'un client en utilisant le protocole SIP se trouve à l'annexe II.

4.5.4 Création d'extensions IAX

Les clients qui se connecteront en utilisant le protocole IAX doivent être positionnés dans le fichier *iax.conf*. Un exemple de configuration d'un client IAX se trouve dans l'annexe III.

4.5.5 Groupes et manipulations des appels

Le fichier *Extension.conf* contient tout le plan de numérotation du serveur. Il permet d'associer les numéros de téléphone (extension) à différentes actions et aussi de définir les groupes et les actions que IPPBX doit faire pour gérer les appels en provenance et à destination de chaque groupe.

Le fichier est composé de deux axes : le contexte global qui contient les variables de portées globales et les contextes particuliers.

Un contexte est une zone de mémoire privée dans laquelle des actions de portée limitée pourront être exécutées.

L'enregistrement d'une extension se fait de la manière suivante :

Exten => extension, Numéro de Séquence, Action;

Le fichier *Extension.conf* contenant le plan de numérotation se trouve à l'annexe IV.

4.6 Interconnexion au réseau PSTN

L'interconnexion de l'IPPBX au réseau téléphonique traditionnel nécessite la carte FXO/FXS. Toutefois, pour pouvoir l'utiliser il faut compiler le progiciel fourni avec la carte Zaptel pour qu'elle soit prise en charge. Les deux commandes nécessaires sont :

```
Modprobe zaptel
Modprobe wcfxo
```

Ensuite, nous ajoutons la ligne suivante dans le fichier `/etc/zaptel.conf` pour déclarer le canal de signalisation et le protocole utilisé :

```
Fxsks = 1 ; la signalisation utilisée est Koolstart dans le canal 1
Defaultzone = us
Loadzone = us
```

La déclaration de signalisation doit être ajoutée dans le fichier `/etc/asterisk/zapata.conf`

```
Signaling = fxs_ks
Group = 1
Channel => 1
```

Finalement, nous configurons un trunk pour que l'interconnexion au réseau PSTN passe par le canal 1, et les différents canaux peuvent l'utiliser pour acheminer les données voix ou vidéo. Dans le même fichier utilisé précédemment pour gérer les appels 4.5.5 nous ajoutons un trunk zaptel

```
exten => s,1,Dial,Zap/1
```

4.7 Interconnexion au réseau GSM

Dans cette section, nous décrivons la procédure d'interconnexion IPPBX au réseau GSM mais la fonctionnalité n'est pas testée. Une passerelle GSM est nécessaire pour interconnecter la plateforme IPPBX au réseau GSM, cette passerelle est sous forme de carte PCI, elle dispose de deux emplacements SIM pour permettre deux canaux simultanément vers le réseau GSM.

La configuration de la passerelle GSM est spécifique au fabricant, mais la plateforme IPPBX nécessite l'ajout des lignes suivantes pour pouvoir acheminer les appels vers le réseau GSM.

Dans le fichier GSM, nous ajoutons l'extension suivante pour le réseau GSM :

```
[100]
type = friend
username = 100
password = 100
context = gateway ; le contexte des appels entrants
callerid = la passerelle GSM
host = dynamic
nat = no
allow = ulaw
allow = alaw
```

Ensuite, dans le fichier extensions.conf nous configurons la gestion d'appel comme suit :

```
[gateway]
exten=> _103,1,Answer()
exten => 103,2,Set (TIMEOUT(digit) = 3);
exten => _103,3, (outgoing); le contexte pour acheminer l'appel au réseau GSM

[outgoing]
```

```

exten => _888,1,SetCallerID(«XXXXXX»)
EXTEN => _888,2,Dial(SIP/$EXTEN @103,60,r)

```

Cette configuration ajoutée aux fichiers extensions.conf et sip.conf permettra de faire des appels à partir de clients locaux vers un réseau GSM.

4.8 Interconnexion de deux systèmes IPPBX

Il existe trois manières pour interconnecter les deux systèmes. Premièrement, il s'agissait de la méthode des extensions, mais cette méthode n'est pas utilisable dans le cas où plusieurs canaux souhaitent communiquer simultanément. Dans une autre méthode, l'architecture maître/esclave, le serveur déclaré Peer est maître et peut transférer les appels vers le serveur esclave. Par contre, le serveur User ne peut que recevoir les appels du serveur Peer. Ce concept est utilisable lorsque l'on désire acheminer les appels dans un sens unidirectionnel. Finalement, le modèle Friend/Friend permet aux deux systèmes de communiquer entre eux et d'avoir une transparence totale dans l'acheminement d'appel.

Nous décrivons la configuration du modèle Friend/Friend que nous avons choisie. Nous ajoutons les champs suivants dans le fichier iax.conf :

```

[general]
bindadd = 0.0.0.0
tos = lowdelay
disallow = all
allow = ulaw ; nous utilisons la loi u
allow = g729 ; nous utilisons le codec G.729
register => SiteB :PassSiteB@IP-SiteB ; enregistrement dans le système du site B ;
Nous créons un compte iax pour l'IPBX du site B pour qu'il puisse s'enregistrer sur
l'IPBX A
[SiteA]

```

```

type = friend ; nous utilisons l'arrangement friend
user = SiteA
secret = PassSiteA
host = dynamic
context=From-SiteA
auth=md5 ; nous utilisons l'authentification md5
disallow=all
allow=g729

```

Le type d'authentification est primordial dans ce cas nous utilisons la méthode d'authentification MD5.

Lorsque le fichier `iax.conf` créé dans l'IPPBX A, nous ajoutons un contexte au fichier `extensions.conf`. Nous pouvons laisser une transparence totale entre les deux IPPBs ou nous ajoutons un préfixe pour qu'un appel soit redirigé de l'IPPBX A vers l'IPPBX B. Dans `extensions.conf` nous ajoutons ceci :

```

[To-Site-B]
exten => _22.,1,Dial(IAX/SiteB :PassSiteB@IP-Site-B /${EXTEN :2}) ;

```

Nous utilisons le préfixe 22 pour sortir de l'IPPBX A vers l'IPPBX B. L'appel sera acheminé par le trunk IAX vers l'IPPBX B. `EXTEN : 2` indique à l'IPPBX qu'il faut qu'il ignore les deux premiers chiffres (22).

`exten => _22.,2, !congestion ;` En cas d'échec Asterisk génère une tonalité de congestion.

Dans l'IPPBX B, nous reproduisons la même chose à l'exception de changer dans les fichiers de configurations le "Site B" par "Site A" et changer également le mot de passe.

Le trunk IAX utilise le port 4569, qui doit être ouvert sur les deux systèmes.

Lorsque la configuration est terminée, nous pourrions vérifier le trunk IAX entre les deux IPPBX avec la console d'Asterisk :

La commande *asterisk*CLI> iax2 show peers*

4.9 Conclusion

Ce chapitre montre les recherches antérieures effectuées via la plateforme IPPBX, ainsi que l'installation et la mise en œuvre du système IPPBX. L'interconnexion avec le réseau PSTN et GSM est illustrée également.

Les scripts de configuration des extensions SIP, IAX et le plan de numérotation sont affichés dans les annexes : II, III et IV.

CHAPITRE 5

SIMULATION : SIGNALISATION (SIP/IAX) ET LA QUALITÉ DE LA COMMUNICATION IAX

Ce chapitre décrit la simulation du calcul de temps requis pour la signalisation SIP/IAX en utilisant la plateforme IPPBX, le soft phone Xlite et le capteur de trames WireShark.

Ensuite, nous expliquons l'outil *IAX Call Analyser for Unsniff* utilisé pour tester les performances d'un canal IAX avec les codecs G.711 et GSM dont la bande passante, le délai inter arrivée et la gigue.

5.1 Description de la simulation calcul de signalisation SIP/IAX

Pour pouvoir calculer le temps requis pour la signalisation, nous avons utilisé le soft phone Xlite, le capteur de trames WireShark et IPPBX Asterisk qui a le rôle d'un proxy.

Le temps calculé est basé sur les messages (REGISTER et OK) pour l'enregistrement et (INVITE et OK) pour la signalisation SIP.

Le temps d'enregistrement du client 1 est calculé entre le moment de l'envoi du message REGISTER et la réception du message OK.

Le calcul du temps de signalisation entre deux clients (client 1 et client 2) commence lorsque le client 1 envoie un message INVITE et il se termine lorsque le client 1 reçoit le message OK du client 2. Le capteur de trames WireShark est installé sur la machine du client 1.

L'enregistrement IAX est basé sur les messages (REGREG et REGACK).

La signalisation IAX est calculée en fonction de la durée entre l'envoi du message NEW et la réception de message ACCEPT.

Le temps requis pour l'enregistrement d'un terminal IAX est estimé à 22 ms alors que le protocole SIP nécessite 244 ms.

La signalisation entre deux terminaux IAX est approximativement à 453 ms, les terminaux SIP ont besoin de 697 ms.

Ces résultats sont obtenus à partir de la moyenne de 10 expériences de calcul du temps requis pour l'enregistrement et la signalisation (tableau 5.1). Le capteur de paquets WireShark est utilisé pour réaliser cette tâche.

Tableau 5.1 Les statistiques d'enregistrement et de signalisation SIP/IAX

Expérience	Enregistrement SIP (s)	Enregistrement IAX (s)	Signalisation SIP (s)	Signalisation IAX (s)
1	0.245	0.027	0.766	0.409
2	0.255	0.021	0.727	0.497
3	0.245	0.021	0.679	0.456
4	0.243	0.020	0.721	0.434
5	0.242	0.016	0.700	0.447
6	0.242	0.018	0.672	0.460
7	0.244	0.016	0.682	0.454
8	0.243	0.018	0.639	0.458
9	0.245	0.019	0.708	0.479
10	0.243	0.017	0.685	0.440
Moyenne	0.244	0.022	0.697	0.453

Les résultats obtenus montrent que la signalisation IAX se fait plus rapidement que SIP car la signalisation SIP nécessite « INVITE et OK » pour établir un canal de signalisation et négocier les codecs audio et vidéo.

Le protocole IAX utilise « NEW et ACCEPT » pour signaler et négocier les paramètres média. Le fait que le protocole IAX est binaire donc il nécessite moins de temps d'exécution à l'opposé du protocole SIP, qui est en format texte. Il consomme donc moins de temps machine. En effet, le résultat montre que les deux protocoles SIP et IAX sont sous un seuil acceptable.

Cependant, le temps de signalisation, qu'il soit SIP ou IAX peut affecter les applications temps réel lorsqu'on utilise un protocole de liaison sans fil à bas débit, surtout s'il nécessite la retransmission des paquets de signalisation.

L'article [26] a évalué le temps de signalisation SIP sur une liaison sans fil en utilisant les protocoles UDP et TCP. L'article [33] constate qu'un canal à 9.6 kbps nécessite un temps de signalisation SIP de 84.5 secondes avec le protocole UDP et de 105 secondes avec le protocole TCP.

5.2 La performance d'un canal IAX avec le codec G.711

L'outil *IAX Call Analyser for Unsniff* [41] réalise une analyse complète d'appel IAX, il permet de mesurer et de tracer la bande passante, le délai, et la gigue d'une communication IAX.

Les données capturées pendant l'appel IAX sont stockées dans un fichier pour être analysées ultérieurement.

L'analyseur se base sur les messages ``NEW `` et ``ACCEPT `` pour prendre en compte une communication IAX établie. Les données associées à chaque appel sont extraites pour être analysées.

Pour chaque appel, les données suivantes sont utilisées pour calculer la bande passante, le délai, la gigue et la perte de paquets :

- Numéro d'appel source et l'adresse IP associée.

- Numéro d'appel destination et l'adresse IP associée.
- Début du temps correspond au temps de réception de message ACCEPT
- La durée d'appel
- Le codec utilisé

5.2.1 La méthodologie appliquée pour calculer la bande passante

Lorsqu'un appel est établi, entre deux terminaux IAX, nous exécutons le script de la manière suivante.

Nous capturons les paquets de trafic voix, qui circulent entre les deux terminaux IAX.

Ensuite, nous exécutons le script IAX 2 Call Analyser [41]. Le script utilise un taux d'échantillonnage de 200 ms pour tracer la bande passante utilisée. D'autres échantillonnages peuvent être testés, il suffit de changer le paramètre: @sliceus = 200000 dans le script (voir l'annexe V). Les données sont analysées et affichées dans le graphe de la bande passante.

La figure 5.1 montre la bande passante de la communication. Nous constatons que la bande passante est estimée à 79 kbps avec le codec G.711. La communication utilise le codec G.711 (débit 64 kbps avec une période d'échantillonnage 20 ms).

Chaque paquet émis contient une trame de taille 160 octets :

$(64 \text{ kbps} * 20\text{ms} / 8 \text{ bits par octet} = 160 \text{ octets par paquet})$ et l'en tête IAX/UDP/IP de taille de 32 octets [46]. Donc un paquet de taille 192 octets est généré chaque 20 ms, cela est équivalent à $192 * 8 * 50 \text{ bits/s}$ (76.8kbps).

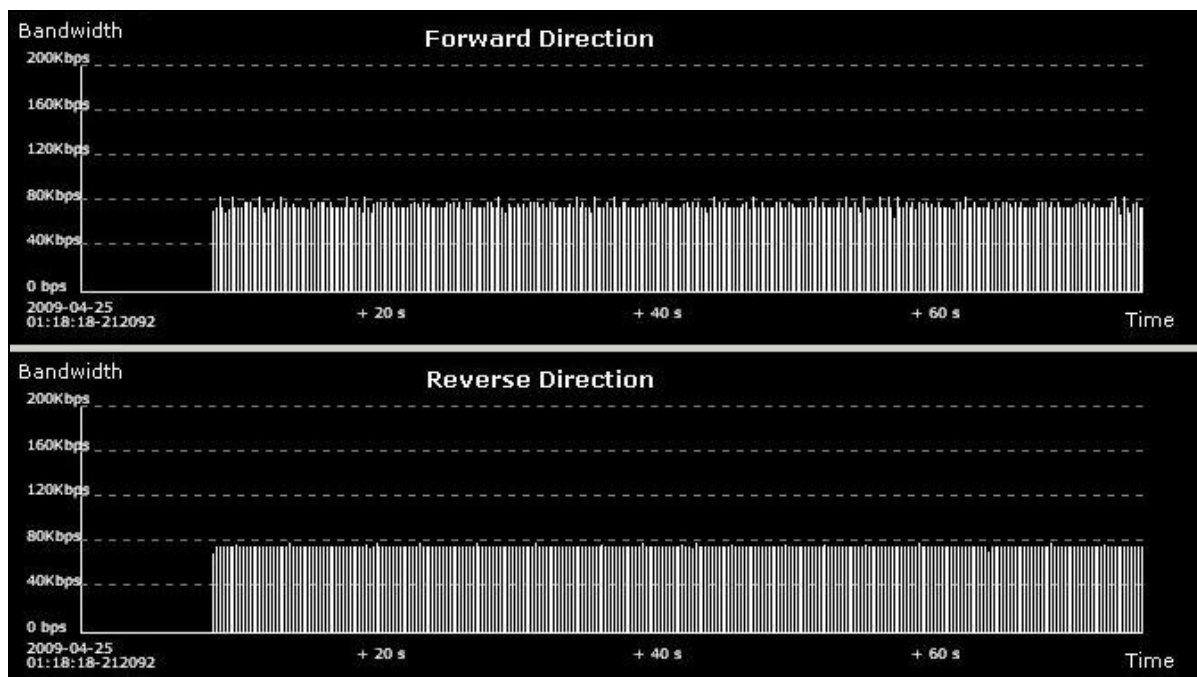


Figure 5.1 La bande passante estimée dans les deux sens.

La bande passante estimée dans les deux sens est à 79 kbps, la valeur obtenue est attendue, car le codec G.711 permet un débit à 64 kbps, et la communication s'est déroulée dans un réseau local.

5.2.2 La méthodologie appliquée pour calculer le délai inter arrivée

L'outil *IAX Call Analyser for Unsniff* [41] permet de calculer le délai inter arrivé de paquets en se basant sur le paramètre TimeStamp de l'entête IAX. La conversion du paramètre TimeStamp en milliseconde à la réception permet de nous indiquer le délai inter arrivé de paquet IAX2 (voir l'annexe VI).

Le délai inter arrivée est calculé à la réception de paquets.

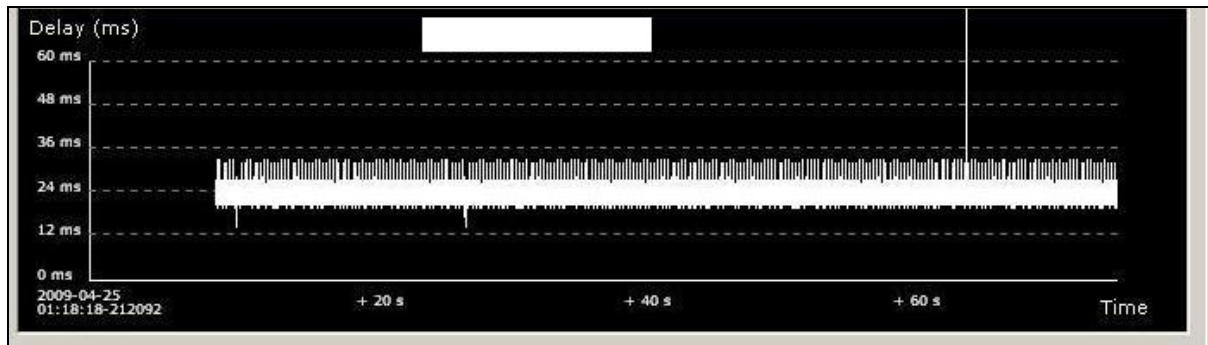


Figure 5.2 Le délai inter arrivée G.711 (ms).

La figure 5.2 montre que le délai de paquets est entre 18 ms et 34 ms. mais on remarque que certains paquets mettent un temps plus élevé pour atteindre la destination. Lorsque le temps est très élevé, ce sont des paquets perdus. On remarque cela très clairement à un peu plus de 60 s à la figure 5.2.

5.2.3 La méthodologie appliquée pour calculer la gigue.

La gigue est la variation causée par le traitement de codage et l'état du réseau. Le temps d'inter-arrivée des paquets est variable sur la durée de la communication. Ce paramètre est un bon indicateur de la qualité d'appel de voix sur IP.

Le draft IAX 2 [34] n'indique pas la formule de calcul de la gigue, l'outil Unsniff utilise la formule du protocole RTP publié dans RFC 3550 [34] :

S_i : L'horodatage du paquet i (TimeStamp).

R_i : L'arrivée du paquet i .

Délai inter-arrivée (Paquet i , Paquet j) = $(R_j - R_i) - (S_j - S_i)$

$|D(i-1, i)| = |(R_i - S_i) - (R_{i-1} - S_{i-1})|$

$J(i) = J(i-1) + (|D(i-1, i)| - J(i-1))/16$

La gigue est calculée dans une seule direction (voir l'annexe VII).

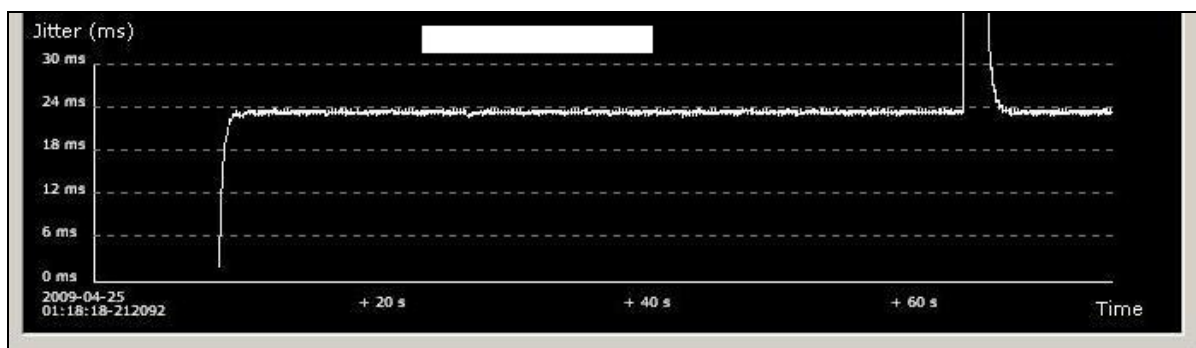


Figure 5.3 La gigue en ms.

La gigue est en moyenne de 24 ms (figure 5.3).

Le résultat de paramètres obtenus : la bande passante = 78 kbps, le délai est entre 18 ms et 34 ms et la gigue = 24 ms nous indique que la qualité de la communication IAX est adéquate car les résultats obtenus sont attendus sachant que la communication s'est déroulée dans un réseau local.

5.3 La performance du canal IAX avec le codec GSM

Nous avons utilisés L'outil *IAX call analyser for Unsniff* [41] pour calculer la bande passante, le délai inter arrivée de paquets et la gigue.

5.3.1 La bande passante GSM

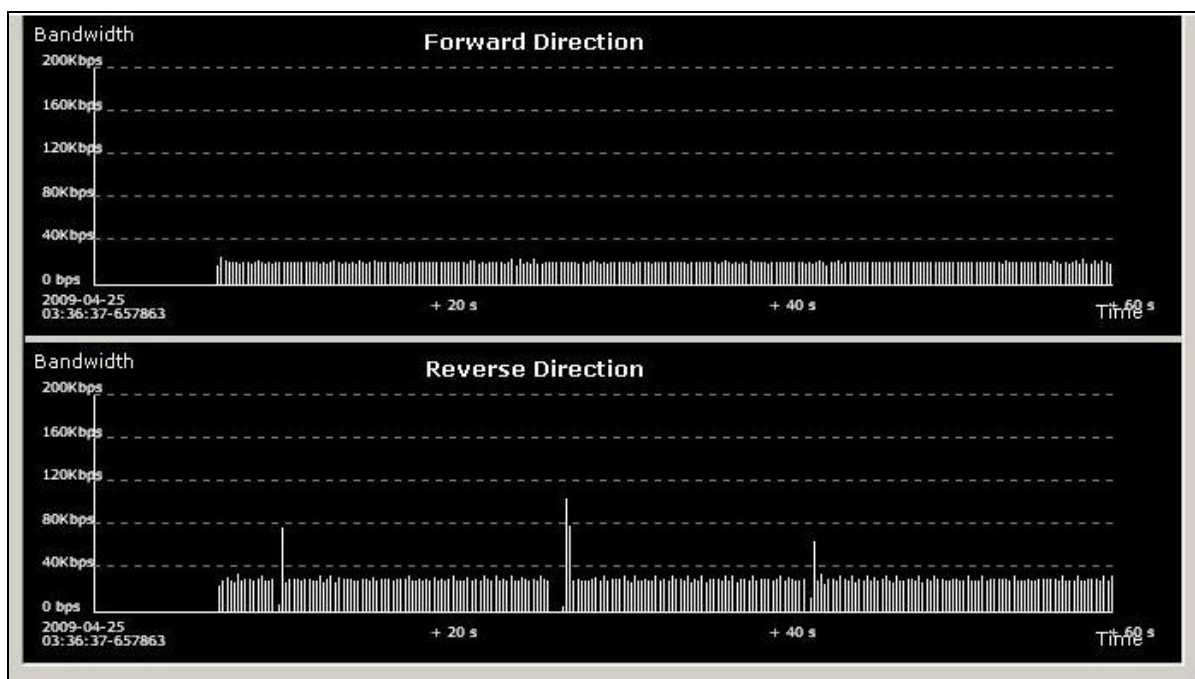


Figure 5.4 La bande passante du canal IAX avec le codec GSM.

La figure 5.4 montre la bande passante estimée du canal à 27 kbps dans les deux sens.

Le codec GSM génère, un échantillon toutes 20 ms à un débit de 13 kbps.

Un échantillon de 20 ms contient une trame de la voix de taille 260 bits, si on ajoute 256 bits de l'entête (IAX/UDP/IP). On obtient la bande passante du codec GSM: 25.8 kbps.

Les résultats expérimentaux sont conformes à la théorie.

À l'inverse du codec G.711, la bande passante de l'appel IAX avec le codec GSM est réduite au détriment de la qualité du codec [46]. MOS(GSM)=3.7 et MOS(G.711)=4.2.

5.3.2 Le délai inter-arrivée GSM

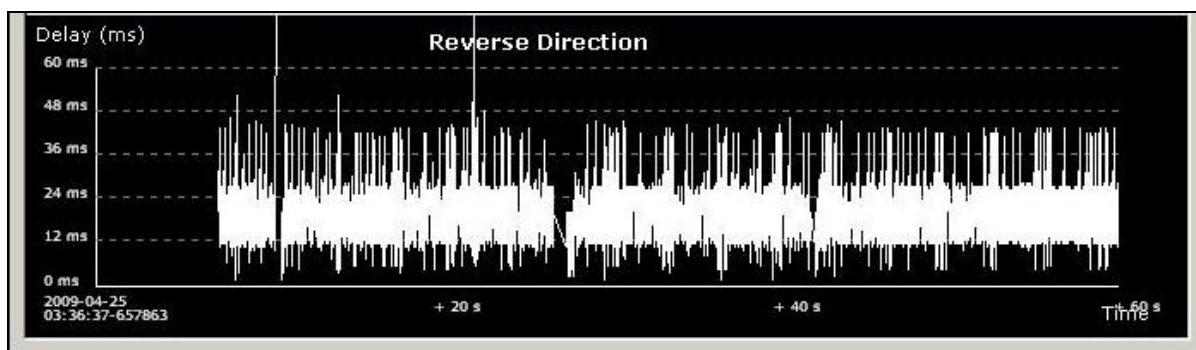


Figure 5.5 Le délai inter arrivée d'une communication IAX avec le codec GSM (ms).

Le délai inter arrivée est un élément de mesure de la qualité au cours d'une communication IAX. En utilisant le codec GSM, il est estimé entre [12, 26 ms] pendant cet appel.

5.3.3 La gigue GSM

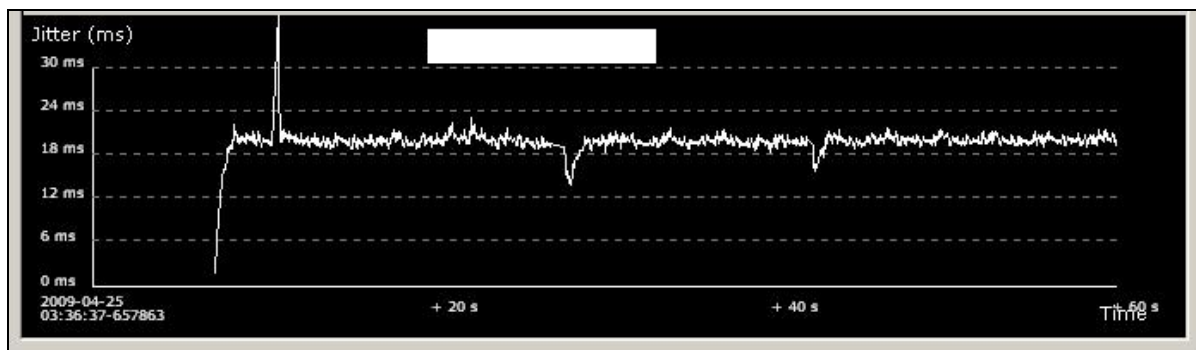


Figure 5.6 La gigue d'une communication IAX avec le codec GSM (ms).

La gigue de l'appel avec le codec GSM est environ 21 ms, la valeur est acceptable pour une communication audible sur le réseau.

5.4 Les événements IAX

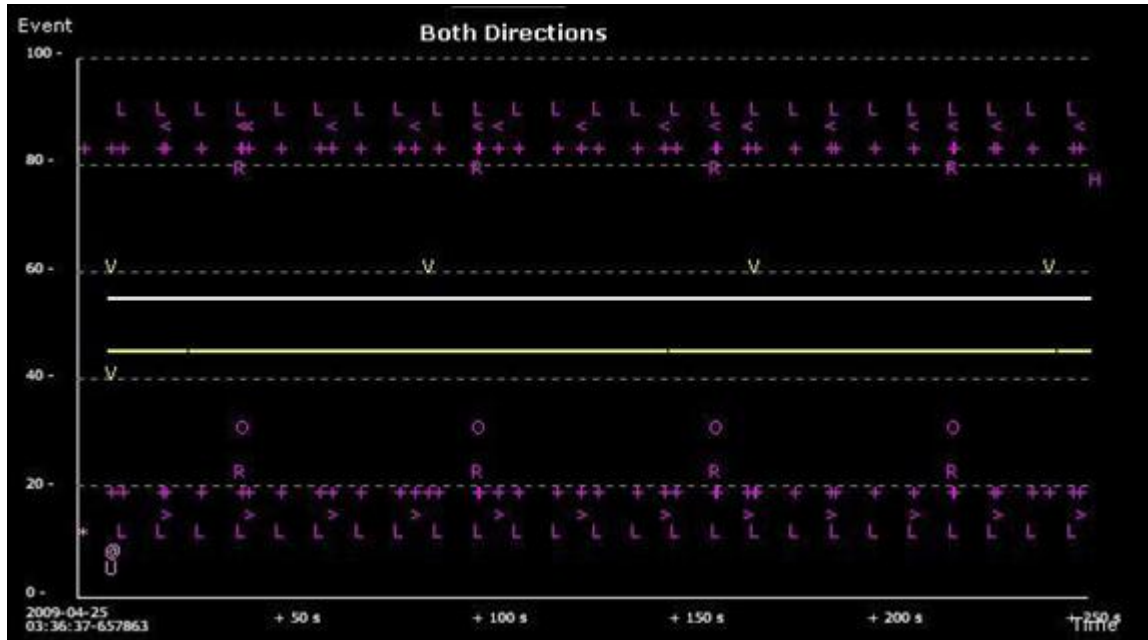


Figure 5.7 Les messages envoyés au cours d'une communication IAX.

La figure 5.7 montre les événements au cours de la communication IAX :

L'axe vertical représente la liste des événements en fonction de la durée d'une communication.

Les messages envoyés au cours de la communication IAX sont représentés par les symboles (U, +, <, >, L, et V). La ligne au dessus de la valeur 40 représente la voix de l'émetteur dans un sens alors que la ligne au dessous de la ligne 60 représente la voix du récepteur dans le sens inverse. Le tableau 5.2 montre la signification de symboles représentatifs de la communication IAX.

On remarque que l'enregistrement est répétitif (symbole R). En fait [34] spécifie une période d'expiration égale à 60 secondes. Après laquelle, le client doit se réenregistrer.

On remarque aussi que les messages PING et PONG (symboles < et >) sont envoyés régulièrement pour maintenir la connectivité entre les deux clients. [34] indique qu'après 20 secondes si l'un des clients ne reçoit pas la charge utile, un message PING est envoyé.

Tableau 5.2 Les événements IAX

Symbole	Signification
R	Message REGISTRATION
+	ACK
N	Message NEW
*	Message RINGING
@	Message ANSWER
A	Message ACCEPT
>	Message PING
<	Message PONG
L	Message LAG (ReQuest et ResPonse)
V	Trame type Full Frame (Voix)
Ligne blanche	Trame type Mini Frame de l'expéditeur
Ligne jaune	Trame type Mini Frame du récepteur

5.5 Conclusion

Les résultats obtenus en calculant le temps de signalisation SIP/IAX dans un contexte local, montrent que la durée de signalisation SIP/IAX est sous un seuil acceptable, et n'a pas d'impact sur la transmission média dans un réseau local. La qualité de la communication IAX en utilisant les codecs G.711 et GSM est adéquate. Les valeurs obtenues du délai et la gigue sont acceptables. Le tableau suivant (5.3) récapitule les résultats de la performance communication IAX en utilisant les codecs G.711 et GSM.

Tableau 5.3 La récapitulation de communication avec les codecs G.711 et GSM

Qualité de service	Codec G.711	Codec GSM
Débit codec	64 kbps	13 kbps
Taille trame générée/20ms	160 octets	32 octets
Bande passante	78 kbps	27 kbps
Délai inter arrivée	[18,34ms]	[12,26ms]
Gigue	24 ms	21 ms

On remarque que le délai inter arrivée de paquets G.711 est plus grand que le délai inter arrivée de paquets GSM. Cela est causé par la taille de trame générée et la bande passante associée pour chaque type du codec.

CHAPITRE 6

IMPLÉMENTATION DU PROTOCOLE RTP DANS UN CLIENT MOBILE DE VIDÉOCONFÉRENCE

Ce chapitre décrit la conception et l'implémentation du protocole RTP dans les terminaux mobiles pour permettre un environnement de vidéoconférence mobile. La technologie utilisée est J2ME associée avec d'autres outils capables de manipuler les flux audio et vidéo en format binaire, et aussi mesurer les performances de l'application.

6.1 Développement d'un client de vidéoconférence mobile

La vidéoconférence mobile est une application enrichissante, car elle permet la transmission du contenu audiovisuel en temps réel. Elle apporte des réductions significatives aux termes de l'argent et du temps à divers secteurs : entreprises, les services hospitaliers et l'assurance.

Par exemple, en utilisant la vidéo mobile, le salarié n'a pas besoin de voyager à travers le monde pour participer à une réunion, le docteur d'urgence peut fournir les instructions paramédicales aux infirmiers d'ambulance.

Actuellement, il existe des solutions propriétaires pour transmettre la vidéo mobile, toutes les technologies utilisées sont spécifiques au constructeur de téléphone mobile. La différence de technologies se catégorise par l'architecture matérielle, les composantes multimédias intégrées, le système d'exploitation et la plateforme de programmation disponible.

La plateforme de développement mobile standard est Java 2 Micro Édition (J2ME), malgré l'existence d'autres alternatives pour les applications mobiles comme Symbian OS (Operating System), Windows Mobile, et Motorola. Ces alternatives peuvent être utilisées pour accomplir notre partie applicative. Nous avons choisi d'utiliser J2ME car la technologie est supportée par la majorité des téléphones mobiles [37].

Notre travail décrit l'implémentation de la vidéoconférence mobile via la technologie J2ME.

L'auteur de [8] a développé une solution VoIP via la technologie J2ME pour envoyer la voix en mode half-duplex.

Vazquez-Briseno et Vincent [9] ont présenté une architecture système adapté aux services streaming mobiles en utilisant un serveur Web.

Le chercheur [10] a expérimenté le streaming en utilisant le protocole (RTSP) en utilisant un serveur RTP/RTSP Darwin pour tester les protocoles RTP et RTSP.

L'idée présentée dans notre partie applicative est différente de [8] et [10]. Nous décrivons l'implémentation du client vidéoconférence mobile en utilisant les différents JSRs (Java Specification Request) et nous ajoutons la fonctionnalité RTP indisponible pour la technologie J2ME.

L'intégration du protocole RTP dans les terminaux mobiles nous permettra d'envoyer et de recevoir les paquets RTP. Le flux vidéo est sous forme des paquets RTP, donc une composante logicielle RTP est nécessaire pour transmettre la voix ou la vidéo sur un terminal mobile.

Notre objectif est de lire le streaming audio/vidéo en temps réel en utilisant (RFC 3261) et (RFC 3550). La partie applicative est organisée comme il suit :

- La description de la technologie utilisée pour capturer et transmettre la voix et vidéo aux téléphones mobiles.
- La solution proposée.
- Les caractéristiques supportées du client vidéoconférence mobile.
- Le résultat de l'implémentation.

6.1.1 La technologie utilisée J2ME

J2ME a été créé afin de résoudre différents problèmes dans le domaine des applications pour les mobiles. Le premier problème est la difficulté pour les programmeurs de développer des applications mobiles. En effet, ils doivent apprendre un langage de programmation et/ou un ensemble d'APIs différent pour chaque terminal mobile ou fabricant (Windows CE, Symbian, etc.). Cela augmente le temps de développement et diminue la motivation et la rentabilité de développer des applications mobiles. J2ME résout ces problèmes en offrant une technologie basée sur le langage Java, très connu des programmeurs, qui se veut portable. Bien évidemment, J2ME offre des fonctionnalités réduites comparées à J2EE afin de refléter les capacités des terminaux mobiles. La disponibilité de la machine virtuelle et JSR sur les téléphones mobiles rend le développement des applications mobiles avec la technologie J2ME plus facile. La portabilité des applications J2ME permet d'éviter l'adaptation du client vidéoconférence mobile à différentes plateformes.

La technologie J2ME utilise JSR (Java Specification Request) pour définir des spécifications et répondre à un besoin dans un domaine bien défini tels que :

- Mobile Multimédia API (JSR 135), cette API est conçue pour gérer la capture et l'affichage du média en temps réel.
- Mobile 3D Graphique API (JSR 134), cette API fournit les fonctionnalités 3D dans un progiciel compact et destiné aux terminaux mobiles.
- File Connection API (JSR 75), définit l'accès aux fichiers système du terminal mobile.
- Web Services API (JSR 172), fournit les moyens nécessaires pour accéder aux services web et aussi traiter un document XML.
- Session Initiation Protocol API (JSR 180), fournit les moyens de communication entre différentes entités. Cette API peut être utilisée pour les messages instantanés, le furetage texte, et les applications vidéoconférence.

Dans ce projet, on utilisera les JSR 180 (SIP) et JSR 135 (multimédia API) [36] pour se connecter au réseau et pouvoir envoyer et recevoir du streaming audio et vidéo. Le protocole de signalisation (SIP) permet au téléphone mobile de se connecter au serveur SIP, et aussi d'être accessible par d'autres utilisateurs dans la même session. Il utilise le protocole SDP pour définir les caractéristiques médias. Les paramètres intéressants pour l'application vidéoconférence mobile sont : le codec média, l'adresse IP, et le numéro du port.

JSR 135 définit une architecture multimédia générique qui contient les composants suivants : lecteur, *Data Source*, et caméra. Le lecteur offre une interface générique pour supporter le type du média, les codecs et le transport. Cependant, les fonctionnalités supportées sont limitées par le constructeur de téléphone mobile. *Data Source* est un objet permettant l'accès aux données média en utilisant un protocole défini, cet objet est capable de localiser et ouvrir une connexion à un fichier multimédia. Chaque objet *Data Source* se compose d'un ou plusieurs flux nommés *Source Stream*.

JSR 135 ne supporte pas le protocole RTP mais il fournit une manière de créer de nouveaux protocoles. Caméra est un nom abstrait d'un composant capable d'accéder et de capturer les données vidéo.

6.1.2 La solution proposée

Ce paragraphe décrit les composantes de la solution proposée :

6.1.3 JSR SIP

Notre implémentation du protocole SIP est basée sur JSR 180 car il nous a permis de créer un module Register pour nous enregistrer au serveur SIP. JSR 180 fournit la méthode *setCredentials(Username, Password, domain)* pour authentifier un téléphone mobile dans un domaine. L'authentification peut se faire pendant la première requête d'enregistrement ou sur une demande du Proxy/Registrar SIP. Le diagramme de transaction des requêtes téléphone mobile, Proxy/Registrar, et le soft phone est illustré dans la figure 6.1.

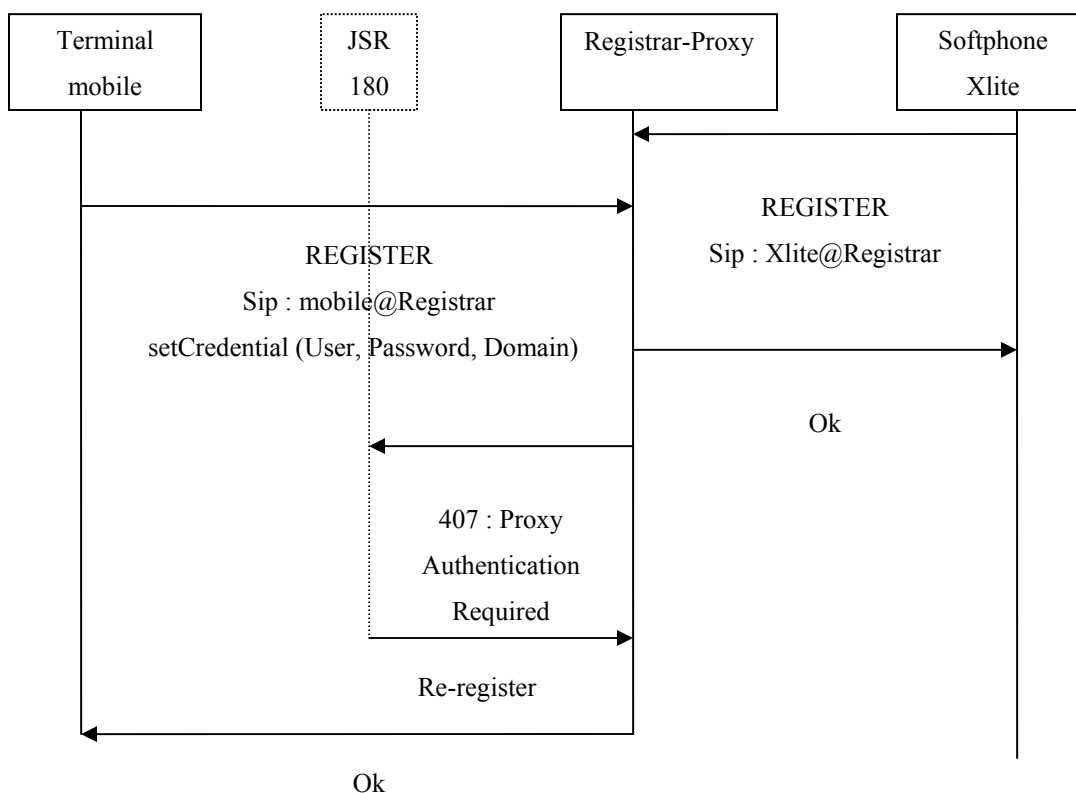


Figure 6.1 Les transactions de requêtes : mobile, proxy et soft phone.
Tirée de l'implémentation JSR 180 dans les téléphones mobiles [33].

Les messages SIP sont générés en appelant API JSR 180. Le message REGISTER est utilisé pour permettre à l'émulateur de téléphone mobile d'enregistrer son adresse IP et admet également une invitation ultérieure d'autre partie (Soft phone Xlite).

Le protocole SDP est utilisé pour le transport de caractéristiques médias en format texte. Les informations du codec sont obtenues à partir :

- audio.encodings.
- video.encodings.

Ensuite, elles sont affectées au champ M (Media) du protocole SDP.

L'émulateur mobile ne supporte aucun codec à l'exception de données audio et vidéo non compressées (PCM linéaire et RGB), le soft phone Xlite supporte le codec audio G.711 et le codec vidéo H.263 pour transmettre le flux audio et vidéo.

La génération du message INVITE nous oblige à fournir au moins un codec supporté par le soft phone Xlite car il est fondamental d'avoir un codec commun entre les deux parties pour pouvoir établir une session.

Pour notre implémentation, nous avons ajouté le codec audio G.711 et le codec vidéo H.263 au message INVITE pour pouvoir communiquer les deux parties.

6.1.4 JSR 135 Mobile MultiMedia

Le JSR 135 (MMAPI) permet de déterminer les caractéristiques des téléphones mobiles. Ceci est important pour notre application et le support de capture audio et vidéo et les codecs intégrés.

Les paramètres de la configuration multimédia supportés par l'émulateur sont illustrés à la figure 6.2.



Figure 6.2 Les composants multimédias supportés par l'émulateur mobile.

Malgré que le support de capture vidéo soit affiché comme possible, en réalité nous n'avons pas accès à la caméra. Après plusieurs recherches, nous avons constaté que la fonctionnalité est planifiée, mais elle n'est pas implémentée. Nous avons décidé d'utiliser un fichier inclus dans l'API comme source des flux audio et vidéo.

Nous pouvons utiliser la méthode *RecordControl* pour contrôler l'enregistrement des flux audio et vidéo.

L'instance *capturePlayer* pour capturer la vidéo est créée via :

```
Player capturePlayer = Manager.CreatePlayer ("capture:// video")
```

Lorsque nous ne spécifions pas le format vidéo, la première valeur retournée par *support.encoding.video* sera utilisée.

Ensuite, selon le format choisi, les données capturées doivent être stockées dans un tampon avant la fragmentation aux paquets pour le transport.

6.2 Les paquets RTP

La technologie J2ME ne supporte pas l'envoi et la réception de paquets RTP. Cependant, MMAPI offre la possibilité d'inclure d'autres protocoles. Les paquets RTP sont sous forme de datagrammes alors J2ME peut faire une connexion UDP pour les recevoir.

Le flux vidéo est transporté en utilisant le format RTP. Notre implémentation est conçue pour extraire les paquets RTP à partir de datagramme UDP reçus.

La procédure de création des paquets RTP se compose de deux tâches principales :

Tâche 1 : ouverture d'une connexion UDP et réception de datagramme UDP :

- Ouverture d'une connexion au port 33038 pour recevoir les datagrammes UDP.

```
DatagrammeConnection dc = Connector.open("datagram://:33038")
```

- Initialisation d'un tableau qui va contenir les datagrammes UDP

```
byte [] tableau_octet = new byte [longueur]
```

- Création d'un objet Datagramme_UDP

```
Datagram Datagramme_UDP = dc.newDatagram(tableau_octet)
```

- Réception de datagrammes UDP

```
dc.receive(Datagramme_UDP)
```

- Extraire les données de datagrammes Data_UDP

```
byte [] Data_UDP = Datagramme_UDP.getData()
```

Tâche 2 : la mise en œuvre des paquets RTP à partir de datagrammes reçus :

- Initialisation d'un objet RTP_paquet :

```
RTP_paquet {int TypePayload
long SequenceNumber
long TimeStamp
long SSRC
byte [] Payload }
```

- Analyser les données de datagrammes Data_UDP pour mettre en œuvre un paquet RTP.

```
RTP_paquet rtpPQ = new RTP_paquet ();
```

```
int longueur = Data_UDP.getLength() ;
```

```
rtpPQ.set_TypePayload(byte Data_UDP[1])
{
    long TypePayload= Data_UDP[1] & 0xff)
}
```

```
rtpPQ.set_SequenceNumber(Data_UDP[2], Data_UDP[3])
{
    long SequenceNumber = ((Data_UDP[2] << 8 & 0xff) | Data_UDP[3]
& 0xff)
}
```

```
rtpPQ.set_TimeStamp(Data_UDP[4],Data_UDP[5],Data_UDP[6],Data_UDP[7])
{
    long TimeStamp = ((Data_UDP[4] & 0Xff << 24) | (Data_UDP[5] &
```

```

        0Xff << 16) | (Data_UDP[6] & 0Xff << 8) | (Data_UDP[7] & 0Xff))
    }

    rtpPQ.set_SSRC(Data_UDP[8], Data_UDP[9], Data_UDP[10], Data_UDP[11])
    {
        long SSRC = ((Data_UDP[8] & 0Xff << 24) | (Data_UDP[9] & 0Xff << 16) |
        (Data_UDP[10] & 0Xff << 8) | (Data_UDP[11] & 0Xff))
    }

    rtpPQ.set_Payload(Datagramme_UDP)
    {
        int longueur = Datagramme_UDP.getLength
        Payload = new byte (longueur -12)
        for (i=0; i<longueur;i++)
        Payload = Data_UDP[i+12];
    }

```

6.3 Les caractéristiques supportées du client vidéoconférence mobile

L'établissement d'une communication vidéo entre l'émulateur mobile et le soft phone Xlite se fait selon les étapes suivantes :

- S'enregistrer auprès du serveur Proxy/Registrier. L'émulateur mobile est enregistré au Proxy/Registrier en utilisant (RFC 3260) [33]. Les informations d'authentification sont fournies par l'utilisateur. Nous avons testé IPPBX comme Proxy-Registrar. L'enregistrement se fait correctement, mais il existe une incompatibilité au niveau du port négocié entre l'implémentation JSR SIP et IPPBX.

La figure 6.3 montre l'enregistrement de l'émulateur mobile auprès du Proxy/Registrar. Le port d'écoute choisi par le système est : 5060 alors que l'implémentation SIP dans l'émulateur mobile prend en compte le port associé à l'en-tête Contact.

Name/username	Host	Dyn	Nat	ACL	Port	Status
500	(Unspecified)	D	N		0	UNKNOWN
2001/2001	(Unspecified)	D	N		0	UNKNOWN
2000	(Unspecified)	D	N		0	UNKNOWN
200/200	(Unspecified)	D	N		0	UNKNOWN
100			N		5060	UNREACHABLE

Figure 6.3 L'enregistrement de l'émulateur mobile auprès du Proxy/Registrar.

Pour remédier au problème, nous avons utilisé un Proxy/Registrar fournit par l'émulateur J2ME. La figure 6.4 montre que l'enregistrement de l'émulateur mobile auprès du Registrar/Proxy se fait correctement et le message reçu OK valide aussi l'enregistrement. Le paramètre : expires = 36000 fournit dans l'en-tête *Contact* indique que l'enregistrement dure 60 min.

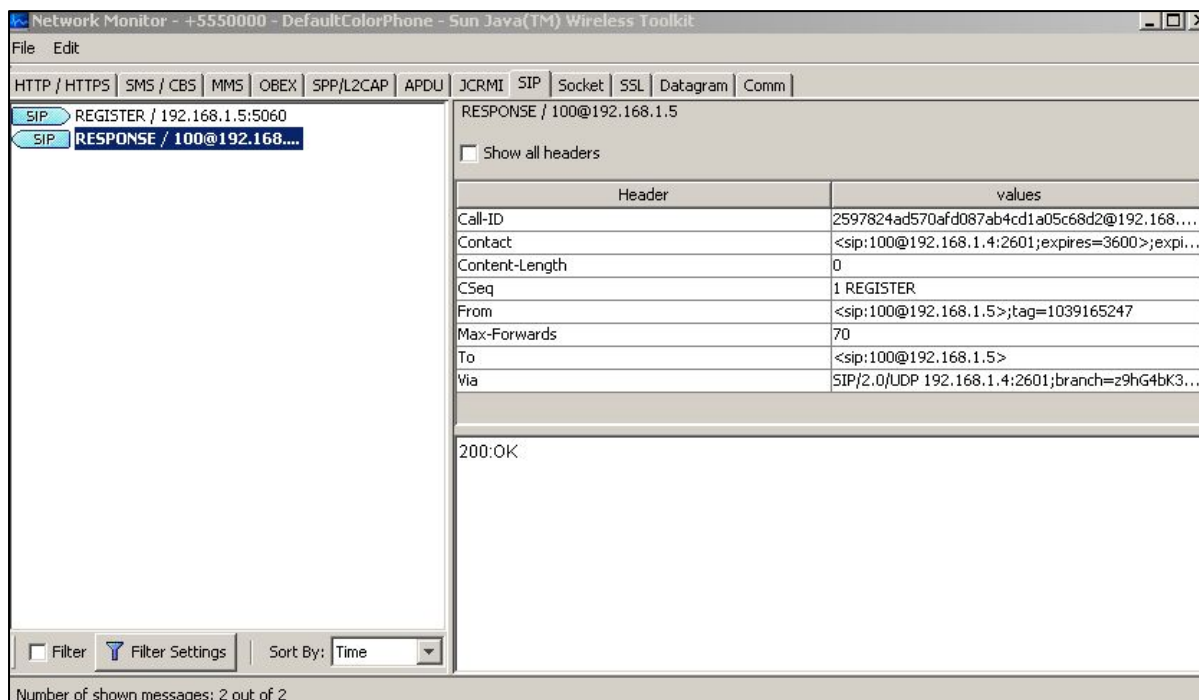


Figure 6.4 L'enregistrement de l'émulateur mobile auprès du proxy SIP (J2ME).

- Déterminer le support de capture voix et vidéo et les codecs associés. Nous pouvons utiliser RecordControl pour lire et stocker les données voix ou vidéo dans un tampon de mémoire. Puis nous pouvons générer les paquets RTP contenant les données média et les en-têtes RTP appropriés. La taille du tampon mémoire ne dépasse pas 128 MB car la taille de téléphone mobile est limitée.
- Inviter le soft phone Xlite à la session. Le message d'invitation utilise le protocole SDP indiquant les paramètres de l'audio et la vidéo. Le codec G.711 loi μ correspond à la valeur 0 au profil RTP, G.711 loi A correspond à la valeur 8 et le codec vidéo H.263 correspond à la valeur 34. La fréquence de l'échantillonnage audio est en général 8000 Hz alors que la fréquence de l'échantillonnage vidéo est 90000 Hz.

La figure 6.5 montre la capture de message INVITE envoyé :

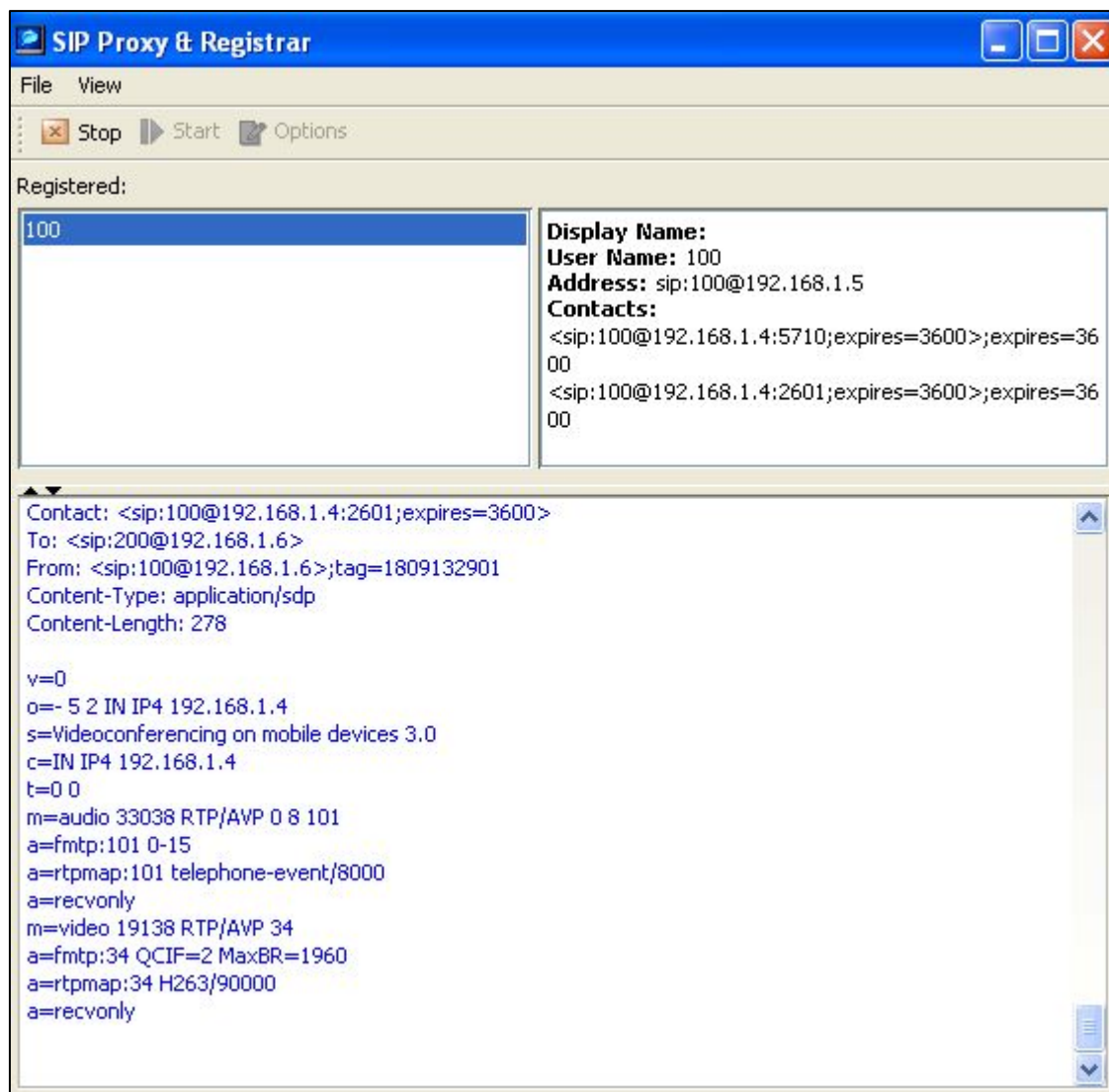


Figure 6.5 Le contenu du protocole SDP.

Dès que l'autre terminal reçoit le message INVITE, il commence à sonner. Cela est traduit par l'envoi du message RINGING. La réponse à l'appel génère le message OK avec les informations nécessaires pour établir la communication audiovisuelle.

L'émulateur mobile reçoit le contenu du protocole SDP et ouvre les canaux pour la réception audio et vidéo (figure 6.5).



Figure 6.6 L'établissement de la communication vidéoconférence.

Nous avons utilisé le capteur de trames WireShark pour analyser ce que nous avons reçu.

No.	Time	Source	Destination	Protocol	Info
40	37.139581	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6479, Time=1752500, Mark
42	37.155220	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6480, Time=1752660
44	37.176155	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6481, Time=1752820
46	37.197192	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6482, Time=1752980
48	37.216539	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6483, Time=1753140
50	37.236253	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6484, Time=1753300
52	37.257093	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6485, Time=1753460
54	37.276868	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6486, Time=1753620
56	37.289100	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6487, Time=1753780
58	37.307376	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6488, Time=1753940
60	37.327177	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6489, Time=1754100
62	37.346853	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6490, Time=1754260
64	37.367927	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6491, Time=1754420
66	37.386673	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6492, Time=1754580
68	37.406287	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6493, Time=1754740
71	37.430087	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6494, Time=1754900
73	37.446189	192.168.1.2	192.168.1.4	RTP	PT=ITU-T G.711 PCMU, SSRC=0x70C8702F, Seq=6495, Time=1755060

[x] Frame 40 (214 bytes on wire, 214 bytes captured) [x] Ethernet II, Src: Vmware_69:75:72 (00:0c:29:69:75:72), Dst: Compalco_fd:0e:09 (00:16:d4:fd:0e:09) [x] Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 192.168.1.4 (192.168.1.4) [x] User Datagram Protocol, Src Port: 46980 (46980), Dst Port: 33038 (33038) [x] Real-Time Transport Protocol					

Figure 6.7 Le flux audio.

Nous remarquons que le codec G.711 est utilisé pour transmettre l'audio. La valeur initiale de TimeStamp de la première trame est aléatoire (*1752500*) *Mark*. (figure 6.7)

Cette valeur augmente à 160 à chaque paquet envoyé, cela représente la durée de mettre en œuvre un paquet RTP d'une durée 20 ms au taux d'échantillonnage 8000 Hz.

Le champ *SSRC = 0x7077C8702F* identifie le flux audio au cours de la communication audiovisuelle.

L'analyse de flux audio indique que la bande passante moyenne est environ 80 kbps, la gigue est estimée à 5 ms. Les valeurs obtenues sont attendues, car l'implémentation s'est déroulée sur une machine dotée de la technologie de virtualisation. Nous avons testé l'implémentation en utilisant l'émulateur mobile et le soft phone Xlite sur le même ordinateur.

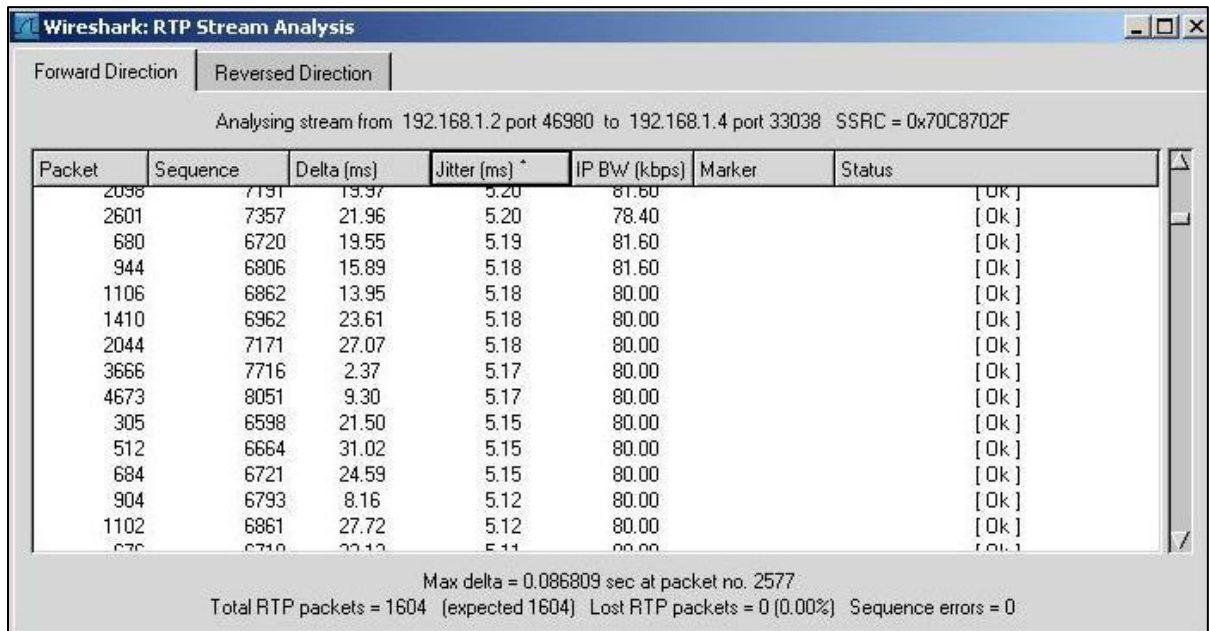


Figure 6.8 Les statistiques du protocole RTP.

L'émulateur mobile a reçu le flux vidéo, le capteur de trames montre le flux vidéo identifié par le champ : SSRC = 0xA08147B6 et utilise le codec H.263 au taux d'échantillonnage 90000 Hz.

No.	Time	Source	Destination	Protocol	Info
257	39.192497	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7072, Time=26679, Mark MODE A H263 payload (PSC) I-Frame
269	39.279266	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7073, Time=29289, Mark MODE A H263 payload (PSC) P-Frame
277	39.344435	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7074, Time=35139, Mark MODE A H263 payload (PSC) P-Frame
283	39.378448	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7075, Time=41079, Mark MODE A H263 payload (PSC) P-Frame
289	39.407847	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7076, Time=44139, Mark MODE A H263 payload (PSC) P-Frame
293	39.443872	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7077, Time=46839, Mark MODE A H263 payload (PSC) P-Frame
299	39.478798	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7078, Time=49899, Mark MODE A H263 payload (PSC) P-Frame
303	39.507979	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7079, Time=53139, Mark MODE A H263 payload (PSC) P-Frame
309	39.544538	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7080, Time=55839, Mark MODE A H263 payload (PSC) P-Frame
315	39.578706	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7081, Time=58989, Mark MODE A H263 payload (PSC) P-Frame
321	39.609510	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7082, Time=62139, Mark MODE A H263 payload (PSC) P-Frame
325	39.643183	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7083, Time=64839, Mark MODE A H263 payload (PSC) P-Frame
331	39.674320	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7084, Time=67989, Mark MODE A H263 payload (PSC) P-Frame
335	39.707148	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7085, Time=70689, Mark MODE A H263 payload (PSC) P-Frame
341	39.743535	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7086, Time=73749, Mark MODE A H263 payload (PSC) P-Frame
349	39.808032	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7087, Time=76989, Mark MODE A H263 payload (PSC) P-Frame
359	39.876124	192.168.1.2	192.168.1.4	H.263	PT=ITU-T H.263, SSRC=0xA08147B6, Seq=7088, Time=82569, Mark MODE A H263 payload (PSC) P-Frame

Figure 6.9 Le flux vidéo.

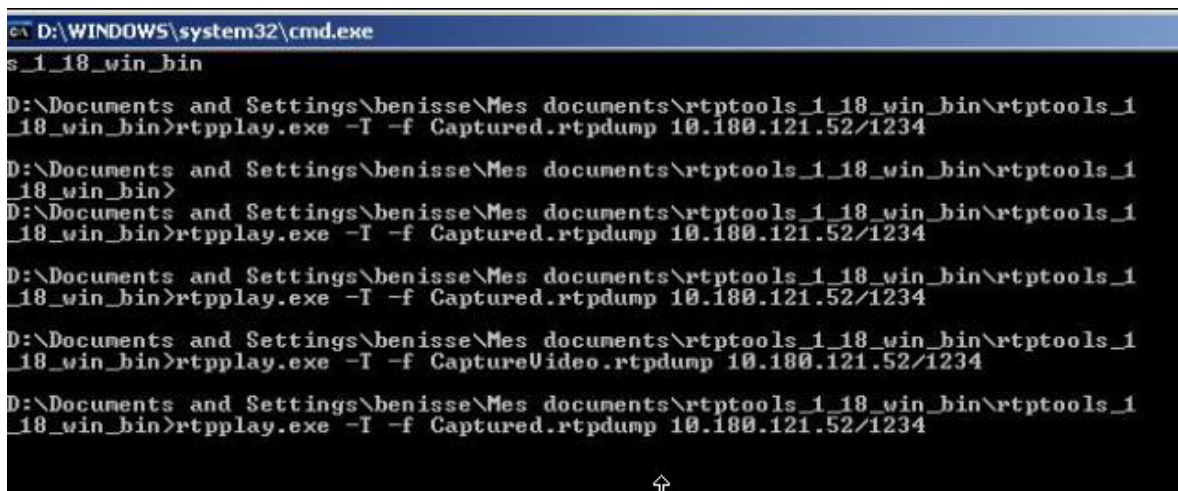
6.4 Résultats de l'implémentation : vidéoconférence mobile

Nous avons enregistré les flux audio et vidéo au format rtpdump pour pouvoir les lire avec l'outil rtpools [47] capable de manipuler les données RTP au format binaire.

rtpplay lit les flux RTP enregistrés au format *rtpdump* à partir d'un fichier.

Ensuite le flux est envoyé à une session RTP définie par l'adresse destination et le port associé.

La figure 6.10 montre la commande utilisée pour rediriger les données binaires à une session RTP.



```

D:\WINDOWS\system32\cmd.exe
s_1_18_win_bin
D:\Documents and Settings\benisse\Mes documents\rtptools_1_18_win_bin\rtptools_1_18_win_bin>rtpplay.exe -I -f Captured.rtpdump 10.180.121.52/1234
D:\Documents and Settings\benisse\Mes documents\rtptools_1_18_win_bin\rtptools_1_18_win_bin>
D:\Documents and Settings\benisse\Mes documents\rtptools_1_18_win_bin\rtptools_1_18_win_bin>rtpplay.exe -I -f Captured.rtpdump 10.180.121.52/1234
D:\Documents and Settings\benisse\Mes documents\rtptools_1_18_win_bin\rtptools_1_18_win_bin>rtpplay.exe -I -f Captured.rtpdump 10.180.121.52/1234
D:\Documents and Settings\benisse\Mes documents\rtptools_1_18_win_bin\rtptools_1_18_win_bin>rtpplay.exe -I -f CaptureVideo.rtpdump 10.180.121.52/1234
D:\Documents and Settings\benisse\Mes documents\rtptools_1_18_win_bin\rtptools_1_18_win_bin>rtpplay.exe -I -f Captured.rtpdump 10.180.121.52/1234

```

Figure 6.10 La manipulation du flux audio et vidéo en mode binaire.

La librairie JMF (Java Media Framework) dotée d'un lecteur JMStudio est capable de lire les flux audio et vidéo.

Nous avons utilisé le lecteur JMStudio pour lire les flux audio et vidéo.

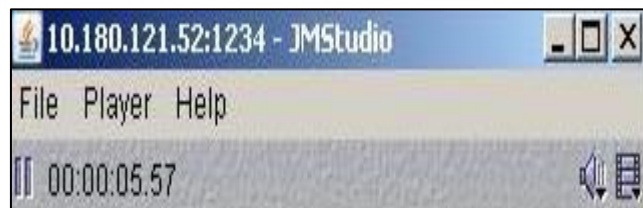


Figure 6.11 La lecture du flux audio.

Le flux audio est effectivement lu par le lecteur audio (figure 6.11).



Figure 6.12 La lecture du flux vidéo.

Le flux vidéo sous forme fichier binaire est également lu par le lecteur vidéo.

L'exécution de l'application client vidéoconférence mobile nécessite les ressources de stockage et de traitement, car la puissance de traitement et de stockage de terminaux mobile est très limitée. La figure 6.13 montre l'utilisation de la mémoire en fonction du temps.

La figure 6.13 montre la taille de mémoire utilisée par l'application (Current), la taille maximale utilisée depuis le lancement de l'application (Maximum)=1.99 Megaoctets, la taille de la mémoire utilisée (Used), la taille de la mémoire inutilisée (Unused) et la taille totale de la mémoire disponible = 2 Megaoctets.

Lorsque le client mobile est en attente des paquets RTP/RTCP, l'application a consommé 1.027 Megaoctet de mémoire. Cela est équivalent à 50% de la capacité totale de mémoire.

La quantité de mémoire fluctue en fonction de nombre de classes appelées et les bibliothèques chargées.

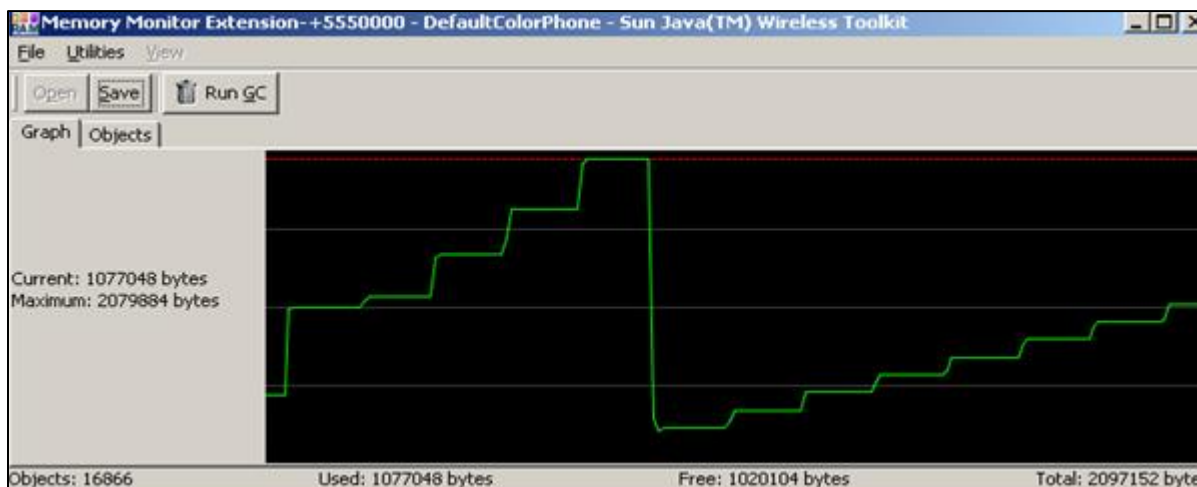


Figure 6.13 La capacité de la mémoire utilisée pendant la session vidéoconférence.

6.5 Discussion de la partie applicative

L'intégration du client mobile supportant la transmission audio et vidéo pour téléphones mobiles a nécessité une étude approfondie de la technologie J2ME.

Le développement d'un client SIP sur les téléphones mobiles capables de s'enregistrer et de signaler n'a posé aucun problème majeur, car l'API SIP offre toutes les classes et les méthodes nécessaires pour réaliser la tâche (voir annexe II).

L'intégration du protocole RTP à partir des datagrammes a demandé une compréhension approfondie du protocole RTP à partir du RFC 3250.

Les résultats obtenus ont démontré que le client mobile de vidéoconférence est capable de joindre une session multimédia et recevoir le flux audio et vidéo.

Les flux sont stockés dans un fichier au format binaire pour être lus par l'outil *rtptools* capable de manipuler les paquets RTP au mode binaire. Le lecteur JMStudio a effectivement lu les flux audio et vidéo.

6.6 Conclusion

Dans ce chapitre, nous avons présenté l'implémentation d'un client de vidéoconférence mobile via la technologie J2ME capable de transmettre la voix et la vidéo à partir de téléphones mobiles. Cette tâche nous a demandé une analyse et une compréhension approfondie de plusieurs protocoles et technologies.

Le protocole SIP est utilisé pour signaler entre le téléphone mobile et le softphone Xlite, la bibliothèque multimédia MMAPI nous a fourni les informations multimédias : caméra, microphone codec audio et vidéo. Celles-ci sont importantes pour négocier les paramètres multimédias en utilisant le protocole SDP.

Le protocole RTP est implémenté grâce à la bibliothèque Network JSR 118 et en utilisant les datagrammes UDP.

Enfin les outils Wireshark et rtpools ont permis d'analyser les médias voix et vidéo reçus.

CONCLUSION

Au début du projet, deux objectifs ont été fixés :

1. Évaluation du délai requis pour l'enregistrement et la signalisation SIP/IAX. Test de performance de communication IAX avec les codecs G.711, GSM.
2. Analyse et conception d'un client J2ME supportant une communication audiovisuelle pendant une session multimédia et incluant l'intégration du protocole RTP.

Dans cette conclusion, chaque objectif sera discuté par rapport à ce que nous avons réalisé :

1. Le premier objectif décrit l'évaluation de l'utilisation du protocole SIP et IAX. Les résultats obtenus à partir des expériences indiquent que le délai requis pour l'enregistrement et la signalisation IAX est plus intéressant que SIP. Le délai de signalisation SIP/IAX peut avoir un impact sur les liaisons à bas débit. La performance a été étudiée par rapport à un seul canal. La qualité de service obtenue indique que la communication IAX en utilisant les codecs G.711 et GSM est telle qu'attendue.
2. Le dernier objectif nous a permis d'approfondir les connaissances de développement des applications mobiles. Nous avons effectué un rapport technique assez complet décrivant la technologie mobile J2ME. Nous avons également testé plusieurs émulateurs et outils capables d'offrir un environnement de vidéoconférence mobile. Le résultat obtenu nous a permis de prouver que le client mobile de vidéoconférence est capable de s'enregistrer auprès d'un Proxy/Registrar pour joindre une session multimédia et de signaler avec d'autres clients de la session via le protocole SIP. La logique de l'intégration du protocole RTP dans la technologie mobile adopte le RFC 3250 sur le plan théorique. L'architecture du système utilisé et les composantes logicielles ont été bien mises en place. La transmission des paquets RTP a été bien réalisée. La manipulation des paquets RTP en mode binaire a été bien effectuée pour rediriger les flux audio et vidéo au lecteur JMStudio.

Cette étude a fait l'objet de deux contributions importantes (1 et 2):

1. L'utilisation de la plateforme IPPBX nous a permis de montrer que c'est un logiciel assez complet et flexible pour tester un système de transmission de la voix ou la vidéo sur IP. La modularité de la plateforme a permis d'examiner plusieurs aspects pendant la transmission média sur le réseau IP (le transcodage, l'état du réseau, le fonctionnement des protocoles, et la sécurité). En revanche, nous avons effectué une évaluation de protocoles SIP/IAX en nous basant sur les statistiques du temps requis pour l'enregistrement et la signalisation. La performance d'une communication IAX en calculant la bande passante, le délai, et la gigue a montré que la voix sur IP en utilisant le protocole IAX sur un réseau local nous offre une qualité normale tel qu'attendu.
2. L'implémentation du client mobile vidéoconférence a montré la faisabilité de l'application vidéoconférence en utilisant la technologie J2ME sur les téléphones mobiles.

RECOMMANDATIONS

Nous recommandons d'effectuer des tests de performance de communication IAX via un simulateur réseau capable de contrôler les paramètres de qualité de service ainsi que de tester plusieurs canaux en parallèle afin d'avoir un résultat plus significatif. L'outil que nous avons utilisé pour analyser la communication IAX, nous a permis de tester les performances pour un seul appel. Nous recommandons aussi de tester les performances du protocole IAX sur un réseau mobile et un réseau distant.

Enfin, le développement de l'application mobile pourra être testé sur un téléphone mobile qui contient tous les composants nécessaires au multimédia.

ANNEXE I

SCRIPT DE COMPILATION LA PLATEFORME IPPBX

```
/* Auteur : Taib */
/*Date : 20 Mai 2008 */
/* Ce programme permet d'automatiser l'installation de la plateforme IPPBX en utilisant les
packages nécessaires*/
#!/usr/bin/sh
# SL Script d'installation Asterisk
#recuperation de la version du kernel d'Asterisk et Zaptel
versionkernel=`cat /proc/version | awk '{print $3}'`
versionasterisk="1.4.4"
versionzaptel="1.4.2.1"
versionlibpri="1.4.0"
versionaddons="1.4.1"

cd /usr/src/
#installation des headers du kernel necessaires a la compilation de zaptel
sh apt-get install kernel-headers-$versionkernel
#recuperation des paquets necessaires a l'installation d'Asterisk
apt-get install zlib1g zlib1g-dev libncurses5 libncurses-dev libssl0.9.6 libssl-dev libnewt-dev
libnewt0.51

#recuperation des archives de zaptel,asterisk,libpri et asterisk-addons
wget http://ftp.digium.com/pub/asterisk/releases/asterisk-$versionasterisk.tar.gz &&
wget http://ftp.digium.com/pub/zaptel/r [...] tel.tar.gz && wget
http://ftp.digium.com/pub/libpri/r [...] sionlibpri && wget http://ftp.digium.com/pub/asterisk
[...] ons.tar.gz

#decompression des archives
```

```
tar -xvzf asterisk-addons-$versionaddons.tar.gz
tar -xvzf asterisk-$versionasterisk.tar.gz
tar -xvzf libpri-$versionlibpri.tar.gz
tar -xvzf zaptel-$versionzaptel.tar.gz

#installation de zaptel , libpri , asterisk , asterisk-addons
cd zaptel-$versionzaptel
make clean
./configure
make
make install
cd libpri-$versionlibpri
make clean
make
make install
cd asterisk-$versionasterisk
make clean
./configure
make
make install
make samples
cd asterisk-addons-$versionaddons
make clean
./configure
make
make install
echo "INSTALLATION D'ASTERISK TERMINEE !!!"
```

ANNEXE II

CONFIGURATION DES EXTENSIONS SIP

/* Auteur : Taib */

/*Date : 20 Mai 2008 */

/* Ce fichier contient les informations de configuration des extensions SIP*/

[1000] ; Chaque client est défini par un label

type = friend ; type de compte associé.

Username = Services clientèle; Nom du client SIP

Secret = 1000; Mot de passe du compte

Record_out = Adhoc; enregistrement des communications sortantes à la demande

Record_in = Adhoc; enregistrement des communications entrantes à la demande

Qualify = yes

Port = 5060; le port d'écoute du protocole SIP

Nat = yes ; activation sur NAT

Mailbox = 1000@default; adresse mail pour l'envoi des messages vocales

Host = dynamic; définition du mode d'Attribution de l'adresse IP

Dtmfmode = rfc 2833 ; type d'envoi des dtmf (les fréquences vocal pour le clavier)

Dial = SIP/1000

Context = from-internal ; tout appel entrant appartient par défaut à la classe from-internal et appliquer les règles de routage dans extensions.conf

Callerid = Service Clientèle P1 <1000> ; identité de l'appelant et numéro d'extension.>

ANNEXE III

CONFIGURATION DES EXTENSIONS IAX

/* Auteur : Taib */

/*Date : 20 Mai 2008 */

/* Ce fichier contient les informations de configuration des extensions IAX */

[1003]

type = friend ; type de compte associé.

Username = Service clientèle P3IAX; Nom du client IAX

Secret = 1003; mot de passe du compte

Record_out = Adhoc; enregistrement des communications sortants à la demande

Record_in = Adhoc; enregistrement des communications entrantes à la demande

Qualify = yes

Port = 4569; port d'écoute du protocole IAX

Mailbox = 1003@default; adresse mail pour l'envoi des messages vocaux

Host = dynamic; définition du mode d'attribution de l'adresse IP

Dial = IAX2/1003

Context = from-internal; permet de relayer vers les règles de routage dans extensions.conf

Callerid = service clientèle P3 <1003>; identité de l'appelant et numéro d'extension.

>

ANNEXE IV

CONFIGURATION DU PLAN DE NUMÉROTATION

```
/* Auteur : Taib */
/*Date : 20 Mai 2008 */
/* Ce fichier contient les informations de configuration du plan de numérotation */
[ext-local]
include =>ext-local-custom
exten=>1000, 1, Macro(exten-vm, 1000, 1000)
exten=>1000, n, hangup
exten=>1000,hint, SIP/1000
exten =>${VM_PREFIX} 1000, 1, Macro(vm, 1000, DIRECTDIAL)
exten => 1000,1,Macro(exten-vm,1000,1000)
exten => 1000,n,Hangup
exten => 1000,hint,SIP/1000
exten => ${VM_PREFIX} 1000,1,Macro(vm,1000,DIRECTDIAL)
exten => ${VM_PREFIX} 1000,n,Hangup
exten => 1001,1,Macro (exten-vm,1001,1001)
exten => 1001,n,Hangup
exten => 1001,hint,SIP/1001
exten => ${VM_PREFIX} 1001,1,Macro(vm,1001,DIRECTDIAL)
exten => ${VM_PREFIX} 1001,n,Hangup
exten => 1002,1,Macro(exten-vm,1002,1002)
exten => 1002, n, Hangup
exten => 1002, hint, SIP/1002
exten => ${VM_PREFIX} 1002, 1, Macro(vm,1002,DIRECTDIAL)
  exten => ${VM_PREFIX} 1002,n,Hangup
>
```

ANNEXE V

MODULE DU CALCUL BANDE PASSANTE IAX

Le code source suivant est publié sur le site : www.unleashnetworks.com

```
# CallBandwidth
# bandwidth used by the codec of this call in a direction
class CallBandwidthModel < ModelBase
  def initialize(call, legno, filename)
    super(call,legno,filename)
  end

  ef each_val
    unsniffDB = WIN32OLE.new("Unsniff.Database")
    unsniffDB.OpenForRead(@captureFile)
    allPackets = unsniffDB.PacketIndex
    print "pid start = #{@activeCall.startPacketID} to #{@activeCall.endPacketID}\n"
    byteCount = 0
    currSecs = Time.at(@activeCall.startTimeSecs,@activeCall.startTimeUSecs)
    (@activeCall.startPacketID..@activeCall.endPacketID-1).each do |pid|
      packet = allPackets.Item(pid)
      @chartFlag = CHART_FLAG_NORMAL
      iaxlayer = packet.FindLayer("IAX2")
      if iaxlayer && matchCall(packet,iaxlayer)
        pkttime = Time.at(packet.TimestampSecs, packet.TimestampUSecs)
        diff = pkttime - currSecs
        if diff*USEC_PER_SEC < @sliceus
          byteCount += packet.Length
        else
          case @chartFlag
```

```

when CHART_FLAG_NORMAL
  yield          pkttime,          8*(byteCount          /          diff),
  DataPointStyle.new(DataPointStyle::LINEHEIGHT,"o")
when CHART_FLAG_VOICE  yield  pkttime,  8*(byteCount  /  diff),
  DataPointStyle.new(DataPointStyle::POINTCHAR,"V",FXRGB(0,0,255))
when CHART_FLAG_CONTROL
  yield          pkttime,          8*(byteCount          /          diff),
  DataPointStyle.new(DataPointStyle::POINTCHAR,"C",FXRGB(255,0,255))
end
byteCount = 0
currSecs = pkttime
end
#-----
# Trunk Call Bandwidth
# bandwidth used by the codec of this call in a direction
class TrunkCallBandwidthModel < ModelBase
  def initialize(call, legno, filename)
    super(call,legno,filename)
  end
  def each_val
    unsniffDB = WIN32OLE.new("Unsniff.Database")
    unsniffDB.OpenForRead(@captureFile)
    allPackets = unsniffDB.PacketIndex
    print "pid start = #{@activeCall.startPacketID} to #{@activeCall.endPacketID}\n"
    byteCount =0
    currSecs = Time.at(@activeCall.startTimeSecs,@activeCall.startTimeUSecs)
    (@activeCall.startPacketID..@activeCall.endPacketID-1).each do |pid|

      packet = allPackets.Item(pid)
      @chartFlag = CHART_FLAG_NORMAL
    end
  end
end

```

```
iaxlayer = packet.FindLayer("IAX2")
if iaxlayer && matchCall(packet,iaxlayer)
pkttime = Time.at(packet.TimestampSecs, packet.TimestampUSecs)
diff = pkttime - currSecs
if diff*USEC_PER_SEC < @sliceus
if (lastTrunkCallLength>0)
byteCount += lastTrunkCallLength + 6
else
byteCount += iaxlayer.Size
end>
```

ANNEXE VI

MODULE DU CALCUL DÉLAI IAX

```
# Delay Model
# calculate delay how are recieving packets spaced wrt to IAX2 timestamps
class DelayModel < ModelBase
  @Curr_IAX_Msecs
  @Curr_Capt_Msecs
  def initialize(call, legno, filename)
    super(call,legno,filename)
    @Curr_IAX_Msecs=0
    @Curr_Capt_Msecs=0
  end

  def each_val
    unsniffDB = WIN32OLE.new("Unsniff.Database")
    unsniffDB.OpenForRead(@captureFile)
    allPackets = unsniffDB.PacketIndex
    print "pid start = #{@activeCall.startPacketID} to #{@activeCall.endPacketID}\n"
    byteCount =0
    currSecs = Time.at(@activeCall.startTimeSecs,@activeCall.startTimeUSecs)
    (@activeCall.startPacketID..@activeCall.endPacketID-1).each do |pid|
      packet = allPackets.Item(pid)
      @chartFlag = CHART_FLAG_NORMAL
      iaxlayer = packet.FindLayer("IAX2")
      if iaxlayer && matchCall(packet,iaxlayer)
        if @Curr_Capt_Msecs == 0
          @Curr_IAX_Msecs = getIAXTimestamp(iaxlayer,@Curr_IAX_Msecs)
```

```

@Curr_Capt_Msecs          =      packet.TimestampSecs*MSEC_PER_SEC      +
packet.TimestampUSecs/USEC_PER_MSEC
print "First frame = #{@Curr_IAX_Msecs}\n"
else
newIAXMs = getIAXTimestamp(iaxlayer,@Curr_IAX_Msecs)
newCaptMs          =      packet.TimestampSecs*MSEC_PER_SEC      +
packet.TimestampUSecs/USEC_PER_MSEC
deltaIAX = newIAXMs - @Curr_IAX_Msecs
deltaCapt = newCaptMs - @Curr_Capt_Msecs
delayMs = deltaCapt - deltaIAX
delayMs = -delayMs if delayMs < 0
@Curr_IAX_Msecs = newIAXMs
@Curr_Capt_Msecs = newCaptMs
print "Delay calculated = #{delayMs}\n" if DEBUG_MODE>

```

ANNEXE VII

MODULE DU CALCUL GIGUE

```
# Jitter Model
# calculate jitter as per RFC3550
class JitterModel < ModelBase

  @Curr_IAX_Msecs
  @Curr_Capt_Msecs
  @Curr_Jitter

  def initialize(call, legno, filename)
    super(call,legno,filename)
    @Curr_IAX_Msecs=0
    @Curr_Capt_Msecs=0
    @Curr_Jitter=0
  end

  def each_val
    unsniffDB = WIN32OLE.new("Unsniff.Database")
    unsniffDB.OpenForRead(@captureFile)
    allPackets = unsniffDB.PacketIndex
    print "pid start = #{@activeCall.startPacketID} to #{@activeCall.endPacketID}\n"
    byteCount =0
    currSecs = Time.at(@activeCall.startTimeSecs,@activeCall.startTimeUSecs)
    (@activeCall.startPacketID..@activeCall.endPacketID-1).each do |pid|

      packet = allPackets.Item(pid)
```

```

@chartFlag = CHART_FLAG_NORMAL
iaxlayer = packet.FindLayer("IAX2")
if iaxlayer && matchCall(packet,iaxlayer)
if @Curr_Capt_Msecs == 0
@Curr_IAX_Msecs = getIAXTimestamp(iaxlayer,@Curr_IAX_Msecs)
@Curr_Capt_Msecs      =      packet.TimestampSecs*MSEC_PER_SEC      +
packet.TimestampUSecs/USEC_PER_MSEC
print "First frame = #{@Curr_IAX_Msecs}\n"
else
newIAXMs = getIAXTimestamp(iaxlayer,@Curr_IAX_Msecs)
newCaptMs      =      packet.TimestampSecs*MSEC_PER_SEC      +
packet.TimestampUSecs/USEC_PER_MSEC
deltaIAX = newIAXMs - @Curr_IAX_Msecs
deltaCapt = newCaptMs - @Curr_Capt_Msecs
delayMs = deltaCapt - deltaIAX
delayMs = -delayMs if delayMs < 0
@Curr_Jitter += 1.0/16.0*(delayMs - @Curr_Jitter)
@Curr_IAX_Msecs = newIAXMs
@Curr_Capt_Msecs = newCaptMs
print "Jitter calculated = #{@Curr_Jitter}\n" if DEBUG_MODE
>

```

ANNEXE VIII

L'ENREGISTREMENT DU CLIENT MOBILE SIP

/* Auteur : Taib */

/*Date : 20 Avril 2009 */

/* Ce programme permet l'enregistrement du client mobile auprès d'un serveur-proxy SIP en utilisant la technologie J2ME */

import javax.microedition.io.*;

import javax.microedition.midlet.*;

import javax.microedition.lcdui.*;

import javax.microedition.sip.*;

public class RegisterTest extends MIDlet implements CommandListener {

private Display display;

private Form form;

private TextField registrar;

private TextField useraddr;

private TextField contact;

private TextField login;

private TextField password;

private Command sendCmd;

private Command exitCmd;

private String username = "200";

private String password = "200";

private String realm ;

SipConnectionNotifier scn = null;

private String Add;

```

    public RegisterTest() {
String ctaddr = "sip:user@host"; // Contact address
// Open a SipConnectionNotifier in an arbitrary port

try {
    scn = (SipConnectionNotifier) Connector.open("sip:");
    if(scn != null) {
// resolve Contact address from SipConnectionNotifier into
ctaddr = new String("sip:200@"+scn.getLocalAddress()+":"+scn.getLocalPort());

    }
} catch(Exception ex) {
    ex.printStackTrace();
}

display=Display.getDisplay(this);
form = new Form("Register to Server");
registrar = new TextField("Registrar address:", "sip:", 40, TextField.LAYOUT_LEFT);
login = new TextField("Login :", "200", 10, TextField.LAYOUT_LEFT);
password = new TextField("Password:", "****", 10, TextField.PASSWORD);
useraddr    =    new    TextField("From-To:",    "sip:"+login.getString(),    40,
TextField.LAYOUT_LEFT);
contact = new TextField("contact:", ctaddr, 30, TextField.LAYOUT_LEFT);
form.append(registrar);
    form.append(login);
    form.append(password);
form.append(useraddr);
form.append(contact);
sendCmd = new Command("Register", Command.ITEM, 1);
form.addCommand(sendCmd);

```

```

exitCmd = new Command("Exit", Command.EXIT, 1);
form.addCommand(exitCmd);
form.setCommandListener(this);
    }

    public void commandAction(Command c, Displayable d) {
if(c == sendCmd) {
    Thread t = new Thread() {
        public void run() {
register();
        }
    };
    t.start();

}
if(c == exitCmd) {
    destroyApp(true);
    }

}

    public void startApp() {
display.setCurrent(form);

}

    public void register() {
        int scode;
        boolean handl= false;
try {

```

```

SipClientConnection scc = null;
// Open a SipClientConnection for REGISTER targeting the
// registrar address
scc = (SipClientConnection) Connector.open(registrar.getString());
scc.initRequest("REGISTER", scn);
// Set necessary headers
scc.setHeader("From", useraddr.getString());
scc.setHeader("To", useraddr.getString());
scc.setHeader("Contact", contact.getString());
    // Send it out
scc.send();

```

```

while(!handl) {
    SipHeader sh;
    // wait max 30 secs for response
    scc.receive(15000);
    scode = scc.getStatusCode();
    switch(scode)
    {
        case 401:
            sh = new SipHeader("WWW-Authenticate",
                scc.getHeader("WWW-Authenticate"));
            realm = sh.getParameter("realm");
            // strip the quotation marks
            realm = realm.substring(1, realm.length()-1);
            // here for example, prompt user for password
            // for this realm
            // set credentials to initiate re-REGISTER

```

```

        scc.setCredentials(login.getString(), password.getString(), realm);
        break;

    case 407:
        sh = new SipHeader("Proxy-Authenticate", scc.getHeader("Proxy-Authenticate"));
        realm = sh.getParameter("realm");
        // strip the quotation marks
        realm = realm.substring(1, realm.length()-1);

        // here for example, prompt user for password
        // for this realm
        // set credentials to initiate re-REGISTER
        scc.setCredentials(login.getString(), password.getString(), realm);
        break;

    case 200:
        // handle OK response
        form.append(new StringItem("Mobile phone is", "registered"));
        handl = true;
        break;

    default:
        // handle other responses
        handl = true;
        form.append(new StringItem("Mobile phone is", scc.getReasonPhrase()));
    }
}

} catch(Exception ex) {

```

```
        ex.printStackTrace();
    }
}

public void pauseApp() {
}

public void destroyApp(boolean b) {
    notifyDestroyed();
}

}>
```

ANNEXE IX

LA SIGNALISATION DU CLIENT MOBILE SIP

/* Auteur : Taib */

/*Date : 20 Avril 2009 */

/* Ce module utilise la bibliothèque JSR-180 (le protocole SIP) pour signaler et négocier les paramètres d'une session multimédia en utilisant les protocoles SIP et SDP.

package InviteSIP;

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import javax.microedition.io.*;
import javax.microedition.sip.*;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.Datagram;
import javax.microedition.io.Connector;
import javax.microedition.media.Control;
import javax.microedition.io.SocketConnection;
import javax.microedition.io.DatagramConnection;
import javax.microedition.media.protocol.SourceStream;
import javax.microedition.media.protocol.ContentDescriptor;
```

/**

* @author taib

```

*/
public class EnvoyerInvite extends MIDlet implements
    CommandListener, SipClientConnectionListener, SipServerConnectionListener {

    private Display display;
    private long startTime;
    private Form form;
    private TextField address;
    private Command startCmd;
    private Command restartCmd;
    private Command byeCmd;
    private Command exitCmd;
    private SipDialog dialog;
    private StringItem str;

    //States
    private final short S_OFFLINE = 0;
    private final short S_CALLING = 1;
    private final short S_RINGING = 2;
    private final short S_ONLINE = 3;

    //Protocol constants - headers:
    private final String FROM_HEADER = "From";
    private final String TO_HEADER = "To";
    private final String CONTACT_HEADER = "Contact";
    private final String CONTENT_LENGTH_HEADER = "Content-Length";
    private final String CONTENT_TYPE_HEADER = "Content-Type";
    private final String ACCEPT_CONTACT_HEADER = "Accept-Contact";
    private final String CALL_ID_HEADER = "Call-ID";
    //Protocol constants - some header values

```

```

private final String SDP_MIME_TYPE = "application/sdp";
private final String ACCEPT_CONTACT = "*;type=\"\" +
SDP_MIME_TYPE + "\"";
private final String DESTINATION_SIP_URI = "sip:200@192.168.1.6";
//Protocol constants - status
private final int OK_STATUS = 200;
private final int RING_STATUS = 180;
private final int UNSUCCESS_STATUS = 400;
private final int METHOD_NOT_ALLOWED_STATUS = 405;
//Protocol constants - methods
private final String INVITE_METHOD = "INVITE";
private final String PRACK_METHOD = "PRACK";
private final String BYE_METHOD = "BYE";
//Timeouts values
private final int TIME_OUT = 30000;
private final int RECEIVE_TIMEOUT = 15000;
//UI labels and messages
private final String INVITE_LABEL = "INVITE";
private final String BYE_COMMAND_LABEL = "Hang-up";
private final String FORM_LABEL = "Videoconference mobile";
private final String RESTART_CMD_LABEL = "Restart";
private final String START_CMD_LABEL = "Call...";
private final String EXIT_CMD_LABEL = "Exit";
private final String SCC_RES = "Response: ";
private final String RING = "RINGING...\n";
private final String DIALOG_LABEL = "Early-Dialog state: ";
private final String DIALOG_STATE = "Dialog state: ";
private final String SESSION_ESTABLISHED = "Session established: ";
private final String SESSION_FAIL = "Session failed: ";
private final String NO_ANSWER = "No answer: ";

```

```

private final String HANG_UP = "user hang-up: ";
private final String SESSION_CLOSE_NOTIFY = "Session closed successfully...";
private final String ERROR = "Error: ";
private final String EXCEPTION = "Exception: ";
private final String NO_DIALOG_INFO = "No dialog information!";
private final String SESSION_CANCEL = "Session canceled: ";
private final String ERROR_CANCEL = "Error canceling the call...";
private final String OTHER_HANG_UP = "Other side hang-up!";
private final String CLOSE_NOTIFIER = "Closing notifier...";

private short state = S_OFFLINE;
private SipConnectionNotifier scn;
private SipServerConnection ssc = null;
private SipClientConnection scc = null;
private String contact = null;
RecevoirRTP recRTP;

// using static SDP content as an example
//private String sdp = "v=0\no=sippy 2890844730 2890844732 IN IP4
host.example.com\ns=example code\nc=IN IP4 host.example.com\nt=0 0\nm=message
54344 SIP/TCP\na=user:sippy";
/*private String sdp = "v=0\no= Emulator Mobile 2890844732 IN IP4 192.168.1.2\n" +
"s=Videoconferencing Mobile\nc=IN IP4 192.168.1.2\nt=0 0\n" +
"m=audio 9820 RTP/AVP 0\n" +
"a=recvonly \n";*/

//private String sdp = "\nv=0\no=100 IN IP4 192.168.1.4\ns=Mobile\nc=IN IP4
192.168.1.4\nt=0 0\n" +
// "m=audio 6000 RTP/AVP 0 8 101\na= fmp:101 0-15\na= rtpmap:0
PCMU/8000\na=recvonly\n";

```

```
//private String sdp = "v=0\no=- 6 2 IN IP4 192.168.1.4\ns=Mobile client\nnc=IN IP4
192.168.1.4\nnt=0 0\nm=audio 39962 RTP/AVP 0 8 101\na=fmtp:101 0-15\na=rtpmap:101
telephone-event/8000\na=recvonly\n";
```

```
private String sdp = "v=0\no=- 5 2 IN IP4 192.168.1.4\ns=Videoconferencing on mobile
devices 3.0\nnc=IN IP4 192.168.1.4\nnt=0 0\n" +
```

```
    "m=audio 33038 RTP/AVP 0 8 101\n" +
```

```
    "a=fmtp:101 0-15\n" +
```

```
    "a=rtpmap:101 telephone-event/8000\na=recvonly\n" +
```

```
    "m=video 19138 RTP/AVP 34\na=fmtp:34 QCIF=2 MaxBR=1960\na=rtpmap:34
H263/90000\na=recvonly\n";
```

```
public EnvoyerInvite() {
```

```
    // Initialize MIDlet display
```

```
    display=Display.getDisplay(this);
```

```
    // create a Form for progress info printings
```

```
    form = new Form(FORM_LABEL);
```

```
    address = new TextField(INVITE_LABEL + ": ", DESTINATION_SIP_URI,40,
TextField.LAYOUT_LEFT);
```

```
    form.append(address);
```

```
    byeCmd = new Command(BYE_COMMAND_LABEL, Command.CANCEL, 1);
```

```
    restartCmd = new Command(RESTART_CMD_LABEL, Command.OK, 1);
```

```
    startCmd = new Command(START_CMD_LABEL, Command.OK, 1);
```

```
    form.addCommand(startCmd);
```

```
    exitCmd = new Command(EXIT_CMD_LABEL, Command.EXIT, 1);
```

```
    form.addCommand(exitCmd);
```

```
    form.setCommandListener(this);
```

```
}
```

```
public void startApp() {
```

```
    display.setCurrent(form);
```

```
}

public void pauseApp() {

}

public void destroyApp(boolean b) {
    notifyDestroyed();
}

public void shutdown() {
    destroyApp(false);
}

public void commandAction(Command c, Displayable d) {
    if(c == startCmd) {
        form.deleteAll();
        form.removeCommand(startCmd);
        form.addCommand(byeCmd);
        state = S_CALLING;
        Thread t = new Thread() {
            public void run() {
                startSession();
            }
        };
        t.start();
        return;
    }

    else if(c == exitCmd) {
```

```

        destroyApp(true);
        return;
    }

    else if(c == byeCmd) {
        if(state == S_RINGING) {
            sendCANCEL();
        } else {
            sendBYE();
        }
        form.removeCommand(byeCmd);
        form.addCommand(restartCmd);
        return;
    }

    else if(c == restartCmd) {
        stopListener();
        form.removeCommand(restartCmd);
        form.addCommand(startCmd);
        form.deleteAll();
        form.append(address);
        return;
    }
}

private void startListener() {
    SipClientConnection regObj = null;
    try {
        if(scn != null)
            scn.close();
    }
}

```

```

        scn = (SipConnectionNotifier) Connector.open("sip.*;type=\"\" + SDP_MIME_TYPE +
        \"\");
        scn.setListener(this);
        contact = new String("sip:100@" + scn.getLocalAddress() + ":" +
        scn.getLocalPort()+";expires=3600");

```

```

        regObj = (SipClientConnection) Connector.open("sip:192.168.1.5:5060");
        regObj.initRequest("REGISTER", scn);
        regObj.setHeader("Contact",contact );
        regObj.setHeader("To", "sip:100@192.168.1.5");
        regObj.setHeader("From", "sip:100@192.168.1.5");

```

```

        regObj.send();

```

```

        boolean flag = regObj.receive(200000);
        if(false == flag)
            throw new RuntimeException("Response not arrived");
        if(regObj.getStatusCode() != 200)
            throw new RuntimeException("status code " + regObj.getStatusCode());
        regObj.close();

```

```

    } catch(IOException ex) {
        form.append(EXCEPTION + ex.getMessage());
    }
}

```

```

private void startSession() {
    try {
        state = S_CALLING;
        // start a listener for incoming requests
        startListener();
        form.append("Videoconferencing mobile client is registered.");
        // SIP connection with remote user
        scc = (SipClientConnection) Connector.open(address.getString());
        scc.setListener(this);
        // initialize INVITE request
        scc.initRequest(INVITE_METHOD, scn);

        //scc.setHeader(ACCEPT_CONTACT_HEADER, ACCEPT_CONTACT);
        scc.setHeader(CONTACT_HEADER, contact);
        scc.setHeader("To", "sip:200@192.168.1.6");
        scc.setHeader("From", "sip:100@192.168.1.6");
        scc.setHeader(CONTENT_TYPE_HEADER, SDP_MIME_TYPE);
        scc.setHeader(CONTENT_LENGTH_HEADER, ""+sdp.length());
        OutputStream os = scc.openContentOutputStream();
        os.write(sdp.getBytes());
        os.close(); // close and send
        str = new StringItem(INVITE_LABEL + "...", scc.getHeader(TO_HEADER));
        form.append(str);
    }
}

```

```

        } catch(Exception ex) {
            ex.printStackTrace();
            form.append(EXCEPTION + ex.getMessage());
        }
    }
}

/**
 * Processes incoming responses sent from the other party based on its
 * contents.
 * @param SipClientConnection the object represents a SIP client
 * transaction.
 */
public void notifyResponse(SipClientConnection scc) {
    int statusCode = 0;
    boolean received = false;
    try {
        scc.receive(20000);
        // fetch response

        state = S_OFFLINE;
        statusCode = scc.getStatusCode();
        str = new StringItem(SCC_RES,statusCode+" "+scc.getReasonPhrase());
        form.append(str);
        if(statusCode < OK_STATUS) {
            dialog = scc.getDialog();
            //form.append(DIALOG_LABEL+dialog.getState()+"\n");
        }
        if(statusCode == RING_STATUS){
            state = S_RINGING;
            form.append(RING);
        }
    }
}

```

```

    }
    if(statusCode == OK_STATUS) {
        String contentType = scc.getHeader(CONTENT_TYPE_HEADER);
        String contentLength = scc.getHeader(CONTENT_LENGTH_HEADER);

        int length = Integer.parseInt(contentLength);

        if(contentType.equals(SDP_MIME_TYPE)) {

        }

        form.append(DIALOG_STATE+dialog.getState());
        scc.initAck(); // initialize and send ACK
        scc.setHeader(ACCEPT_CONTACT_HEADER, ACCEPT_CONTACT);
        scc.send();

        form.append("sdp content received.");

        // TRAITEMENT CANAL RTP
        recRTP = new RecevoirRTP ();
        recRTP.start();
        // TRAITEMENT CANAL RTP

        dialog = scc.getDialog(); // save dialog info
        str = new StringItem(SESSION_ESTABLISHED,
scc.getHeader(CALL_ID_HEADER));
        form.append(str);

```

```

state = S_OFFLINE;
}else if(statusCode >= UNSUCCESS_STATUS){
    str = new StringItem(SESSION_FAIL,scc.getHeader(CALL_ID_HEADER));
    form.append(str);
    form.removeCommand(byeCmd);
    form.addCommand(restartCmd);
    scc.close();
    state = S_OFFLINE;
}
} catch(IOException ioe) {
    // handle e.g. transaction timeout here
    str = new StringItem(NO_ANSWER, ioe.getMessage());
    form.append(str);
    form.removeCommand(byeCmd);
    form.addCommand(restartCmd);
} catch(Exception e){
}
}

// Ends session with "BYE".
private void sendBYE() {
    if(dialog != null) {
        try {
            SipClientConnection
sc
=dialog.getClientConnection(BYE_METHOD);
            sc.setHeader(ACCEPT_CONTACT_HEADER, ACCEPT_CONTACT);
            sc.send();
            str = new StringItem(HANG_UP, BYE_METHOD + " sent...");
            form.append(str);
            boolean gotit = sc.receive(RECEIVE_TIMEOUT);
            if(gotit) {

```

```

        if(sc.getStatusCode() == OK_STATUS) {
            form.append(SESSION_CLOSE_NOTIFY);
            form.append(DIALOG_STATE + dialog.getState());
        }
        else
            form.append(ERROR + sc.getReasonPhrase());
    }
    sc.close();
    state = S_OFFLINE;
} catch(IOException iox) {
    form.append(EXCEPTION + iox.getMessage());
}
} else {
    form.append(NO_DIALOG_INFO);
}
}

private void sendCANCEL() {
    if(scc != null) {
        try {
            SipClientConnection cancel = scc.initCancel();
            cancel.send();
            if(cancel.receive(TIME_OUT)) {
                str = new StringItem(SESSION_CANCEL,
                    cancel.getReasonPhrase());
                form.append(str);
                state = S_OFFLINE;
            } else {
                form.append(ERROR_CANCEL);
            }
        }
    }
}

```

```

    } catch(Exception ex) {
        ex.printStackTrace();
        form.append(EXCEPTION + ex.getMessage());
    }
}
}

```

```

public void notifyRequest(SipConnectionNotifier scn) {
    try {
        ssc = scn.acceptAndOpen(); // blocking
        if(ssc.getMethod().equals(BYE_METHOD)) {
            // respond 200 OK to BYE
            ssc.initResponse(OK_STATUS);
            ssc.send();
            str = new StringItem(OTHER_HANG_UP, "");
            form.append(str);
            form.append(CLOSE_NOTIFIER);
            form.removeCommand(byeCmd);
            form.addCommand(restartCmd);
            scn.close();
            state = S_OFFLINE;
        } else {
            if(ssc.getMethod().equals(PRACK_METHOD)) {
                ssc.initResponse(OK_STATUS);
                ssc.send();
            } else {
                // 405 Method Not Allowed
                ssc.initResponse(METHOD_NOT_ALLOWED_STATUS);
                ssc.send();
            }
        }
    }
}

```

```

    }
    }
    } catch(IOException ex) {
        form.append(EXCEPTION + ex.getMessage());
    }
}

```

```

private void stopListener() {
    try {
        if(scn != null)
            scn.close();
        scn = null;
    } catch(IOException ex) {
        form.append(EXCEPTION + ex.getMessage());
    }
}

```

```

public class RecevoirRTP extends Thread {
    private DatagramConnection dc = null;
    private boolean exit = false;
    int port_audio = 33038;
    int port_video = 19138;
    RTP_paquet rtpPQ = new RTP_paquet();
    public void run () {
        doReceive (port_audio,port_video);
    }
}

```

```

void doReceive (int port,int port1) {
    Datagram Datagramme_UDP;

    try {

        DatagramConnection dc = (DatagramConnection) Connector.open("datagram://:"+
port);
        DatagramConnection dc_video = (DatagramConnection)
Connector.open("datagram://:"+port1);
        while (! exit){
            form.append("RTP audio canal port:"+port);
            form.append("RTP video canal port:"+port1);
            Datagramme_UDP = dc.newDatagram(dc.getLength());
            dc.receive(Datagramme_UDP);
            byte [] Data_UDP = Datagramme_UDP.getData();
            form.deleteAll();
            form.append("Paquet RTP received");
            System.out.print("Received:"+Data_UDP);
            rtpPQ.set_TypePayload(Data_UDP[1]);
            rtpPQ.set_SequenceNumber(Data_UDP[2], Data_UDP[3]);
            rtpPQ.set_TimeStamp(Data_UDP[4], Data_UDP[5], Data_UDP[6], Data_UDP[7]);
            rtpPQ.set_Payload(Data_UDP);
            String Sniff = rtpPQ.To_String();

            form.append(Sniff);
            System.err.println(Sniff);

        } } catch (Exception e) {

```

```
        System.out.println("Exception: " + e.getMessage());
    if (dc != null) {
        try {

            dc.close();

        } catch (Exception f) {
            System.out.println("Exception: " + f.getMessage());
        }
    }
}
```

ANNEXE X

L'IMPLEMENTATION DU PROTOCOLE RTP

/* Auteur : Taib */

/*Date : Avril 2009 */

/* Ce module fournit une nouvelle manière d'intégrer le protocole RTP en utilisant la technologie J2ME et en se basant sur la bibliothèque Network contenant les datagrammes UDP*/

package InviteSIP;

/** * * @author taib */

public class RTP_paquet {

 long TypePayload;

 long SequenceNumber ;

 long TimeStamp ;

 long SSRC ;

 byte Payload [] ;

 public void set_TypePayload(byte a)

{this.TypePayload= (a & 0xff) ;}

 public void set_SequenceNumber(byte a, byte b)

{

 this.SequenceNumber = ((a << 8 & 0xff) | b & 0xff);

 }

 public void set_TimeStamp(byte a, byte b, byte c, byte d)

```

{
    this.TimeStamp = ((a & 0Xff << 24) | (b & 0Xff << 16) | (c & 0Xff << 8) | (d &
0Xff));
}

public void set_SSRC(byte a, byte b, byte c, byte d)

{
    this.SSRC = ((a & 0Xff << 24) | (b & 0Xff << 16) | (c & 0Xff << 8) | (d &
0Xff));
}

public void set_Payload(byte []Datagramme_UDP)

{int lng = Datagramme_UDP.length -12 ;
this.Payload = new byte [lng];

for (int i=0; i<Datagramme_UDP.length;i++)

this.Payload [i]= Datagramme_UDP[i+12];
}

public long get_TypePayload ()
{ return this.TypePayload;}

public long get_SequenceNumber ()
{return this.SequenceNumber;}

```

```
public long get_TimeStamp ()
    {return this.TimeStamp;}
```

```
public long get_SSRC ()
    {return this.SSRC;}
```

```
public byte [] get_Payload ()
    {return this.Payload;}
```

```
public String To_String ()
    { return
    "RTPPacket " + SequenceNumber +
    ": [" +
    " ssrc=0x" + SSRC +
    ", timestamp=" + TimeStamp +
    ", payload type=" + TypePayload +
    " ]"; }
}
```

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

- [1] Abbasi, Taemoor, Shekhar Prasad, Nabil Seddigh et Ioannis Lambadaris. 2005. « A comparative study of the SIP and IAX VoIP protocols ». In *Canadian Conference on Electrical and Computer Engineering 2005, May 1-4 2005* (Saskatoon, SK, Canada). Vol. 2005, p. 179-183. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States.
- [2] Alam, Md Zaidul, Saugata Bose, Md Mhafuzur Rahman et Mohammad Abdullah Al-Mumin. 2007. « Small office PBX using Voice over internet protocol (VOIP) ». In *9th International Conference on Advanced Communication Technology, ICACT 2007, Feb 12-14 2007* (Gangwon-Do, South Korea). Vol. 3, p. 1618-1622. Institute of Electrical and Electronics Engineers Inc., New York, NY 10016-5997, United States.
- [3] Chava, K. S., et How, J. 2007. « Integration of open source and enterprise IP PBXs ». In *3rd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities - TridentCom '07, 21-23 May 2007* (Lake Buena Vista, FL). p. 70-5. IEEE.
- [4] Chih Yuan, Hung, Tan Chin Ping, Chuang Li Chiung et Lee Wei-Tsong. 2002. « The implementation of the communication framework of SIP and MGCP in VoIP applications ». In 10th IEEE International Conference IEEE, 449-54 p.
- [5] Manjur S Kollar, Anas. F. Bayan, Tat Chee Wan, O.Aboubdalla et Sureswaran.2008 « Control and Media Sessions :IAX with RSW Control Criteria » In International Conference on Nework Applications, Protocols and Services. 2008. p 5
- [6] Floyd, S., et V. Paxson. 2001. « Difficulties in simulating the Internet ». *IEEE/ACM Transactions on Networking*, vol. 9, n° 4, p. 392-403.
- [7] Gokhale, S. et Lu Jijun. 2006. « Signaling performance of SIP based VoIP: a measurement-based approach ». In *GLOBECOM '05. IEEE Global Telecommunications Conference, 28 Nov.-2 Dec. 2005* (St. Louis, MO). p. 5 pp.: IEEE.
- [8] Guo, Fang Mao, Alex Talevski et Elizabeth Chang. 2007. « Voice over Internet protocol on mobile devices ». In *6th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2007; 1st IEEE/ACIS International Workshop on e-Activity, IWEA 2007, Jul 11-13 2007* (Melbourne, VIC, Australia). p. 163-168. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States.

- [9] Mabel Vazquez-Briseno, Pierre Vincent, 2007 "An Adaptable Architecture for Mobile Streaming Applications,"In *IJCSNS International Journal of Computer Science and Network Security*, Vol.7 No.9, September 2007.
- [10] Vikram Goyal, 2006 "Pro Java ME MMAPI: Mobile Media API for Java Micro Edition,"Apress 2006.
- [11] Verma, S., et R. Barnes. 2002. « A QoS architecture to support streaming applications in the mobile environment ». In *5th International Symposium on Wireless Personal Multimedia Communications*, 27-30 Oct. 2002 (Honolulu, HI). Vol. vol.2, p. 514-20. IEEE.
- [12] Yuan, Zhang. 2002. « SIP-based VoIP network and its interworking with the PSTN ». *Electronics & Communication Engineering Journal*, vol. 14, n° 6, p. 273-82.
- [13] Zeadally, S., et F. Siddiqui. 2004. « Design and implementation of a SIP-based VoIP architecture ». In *Proceedings - 18th International Conference on Advanced Information Networking and Applications, AINA 2004, Mar 29-31 2004* (Fukuoka, Japan). Vol. 2, p. 187-190. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, United States.
- [14] Khedr, M., O. A. El Aleem et M. M. Selim. 2007. « Evaluation of SIP-based VOIP in heterogeneous networks ». In *International Conference on Computer Engineering & Systems, 2007. ICCES '07, 27-29 Nov. 2007* (Cairo, Egypt). p. 184-9. IEEE.
- [15] Hersent, Olivier, David Gurle et Jean-Pierre Petit. 2006. *La voix sur IP : déploiement des architectures VoIP, IMS et TISpan, Protocoles SIP 3GPP et IETF, H.323, MGCP*. Paris: Dunod, xviii, 749 p.
- [16] Hojung, Cha, Lee Jongmin, Nang Jongho, Park Sung-Yong, Jeong Jin-Hwan, Yoo Chuck et Choi Jin-Young. 2005. « A video streaming system for mobile phones: practice and experience ». *Wireless Networks*, vol. 11, n° 3, p. 265-74.
- [17] Johnston, Alan B., et Inc Books24x. 2004. *SIP ressource électronique : understanding the Session Initiation Protocol*. Boston, MA: Artech House.
- [18] Kadoch, Michel, et C. A. Université Québec. 2004. *Protocoles et réseaux locaux : l'accès Internet*. Montréal: École de technologie supérieure, xvii, 529 p.
- [19] Kadoch., Michel. 2008. « [Notes de cours MGR-820 Réseau haut débit et nouvelle technologies de IP]. ». Montréal : École de Technologie Supérieure.:
- [20] Lee, K. B., et R. A. Grice. 2006. « The design and development of user interfaces for voice application in mobile devices ». In *2006 IEEE International Professional*

- Communication Conference, 23-25 Oct. 2006* (Saratoga Springs, NY). p. 308-20. IEEE.
- [21] Li, Xiao-jun. 2007. « Design and implementation of VoIP based on SIP ». *Computer Engineering*, vol. 33, n° 18, p. 112-15.
 - [22] Pang, Ai-Chun, Chih-Hsiao Liu, Shu Ping Liu et Hui-Nien Hung. 2005. « A study on SIP session timer for wireless VoIP ». In *IEEE Wireless Communications and Networking Conference, WCNC 2005: Broadband Wirelss for the Masses - Ready for Take-off, Mar 13-17 2005* (New Orleans, LA). Vol. 4, p. 2306-2311. Institute of Electrical and Electronics Engineers Inc., New York, NY 10016-5997, United States.
 - [23] Phillipa Sessini, Anirban Mahanti. 2006. « Observations on Round-Trip Times of TCP Connections ». *Techrepublic*.
 - [24] Sinnreich, Henry, Alan B. Johnston et Inc Books24x. 2006. *Internet communications using SIP ressource électronique : delivering VoIP and multimedia services with Session Initiation Protocol* Hoboken, N.J.: Wiley
 - [25] Stephens, A., et P. J. Cordell. 2001. « SIP and H.323 - interworking VoIP networks ». *BT Technology Journal*, vol. 19, n° 2, p. 119-27.
 - [26] Hanane, Shyame, Ramjee. 2004 « Optimisation of VoIP session setup delay over wireless link using SIP » Center for TeleInfrastruktur, Aalborg University, Niels Jernes vej 12, 9220 Aalborg, Denmark.
 - [27] Curran, Kevin, et Gerard Parr. 2003. « An end to end adaptable architecture for streaming media over IP networks ». *Multimedia Tools and Applications*, vol. 20, n° 3, p. 225-236.
 - [28] J.Rosenberg, H.S., G.Camarillo, A.Johnston, *RFC 3261 SIP: Session Initiation Protocol*. 2002.
 - [29] Schulzrinne, H., *RTP Profile for Audio and Video Conference*. 2003.
 - [30] Clayton, Bradley. 2007. « Securing media streams in an Asterisk-based environment and evaluating the resulting performance cost ». Mémoire en science, Grahamstown, South Africa Rhodes University Johansbeurg, 105 p.
 - [31] Cuetos, Philippe de. 2003. « Network and Content Adaptive Streaming of Layered-Encoded Video over the Internet : Streaming Video over the Internet ». Doctorat en télécom, Paris, École nationale supérieure des télécommunications, 175 p.
 - [32] Curran, K., et G. Parr. 2001. « An adaptable framework for streaming media to mobile applications ». In *Wireless 2001. 13th International Conference on Wireless*

Communications. Proceedings, 9-11 July 2001 (Calgary, Alta., Canada). Vol. vol.2, p. 406-16. TRILabs/Univ. Calgary.

- [33] Bouret, C. 2007, *Java Specification Request 180 : SIP API for J2ME*. 2007.
- [34] Internet Task Force engineering www.ietf.org . En ligne. Consulté le 10 février 2007.
- [35] IPPBX Asterisk www.asterisk.org En ligne. Consulté le 15 janvier 2007.
- [36] Mobile Media API 1.2. Sun. <http://java.sun.com/products/mmapi> En ligne. Consulté le 10 avril 2008.
- [37] J2ME. Official Site <http://java.sun.com/javame/index.jsp> En ligne. Consulté le 12 avril 2008.
- [38] Analyseur du protocole réseau Wireshark, <http://www.wireshark.org>. En ligne. Consulté le 15 janvier 2007.
- [39] Java Specification Requests, <http://jcp.org/en/jsr> En ligne. Consulté le 22 avril 2008.
- [40] Chutet, Marc 2009 www.frameip.com/sondages/sondages_voip.php "Sondage sur la VoIP » En ligne. Consulté le 13 janvier 2007.
- [41] Analyseur du protocole IAX www.unleashnetworks.com En ligne. Consulté le 7 janvier 2007.
- [42] www.voip-info.org En ligne. Consulté le 16 janvier 2007.
- [43] <http://www.cisco.com/en/US/tech/tk652/tk698/> En ligne. Consulté le 20 mars 2008.
« VoIP Per Call Bandwidth consumption »
- [44] <http://www.newport-networks.co.uk/cust-docs/52-VoIP-Bandwidth.pdf>
« VoIP Bandwidth Calculation » En ligne. Consulté le 20 mars 2008.
- [45] <http://www.franceip.fr/bandwidth.html>
« Calcul de bande passante VoIP » En ligne. Consulté le 21 mars 2008.
- [46] http://en.wikipedia.org/wiki/Mean_opinion_score En ligne. Consulté le 6 mars 2009.
- [47] www.cs.columbia.edu/irt/software/rtpptools/ En ligne. Consulté le 12 mars 2009.
- [48] www.conutерpath.com En ligne. Consulté le 14 juin 2008.
- [49] www.ethereal.com En ligne. Consulté le 15 mars 2009
- [50] <http://www.iaxtalk.com/iaxlite.html> En ligne. Consulté le 10 avril 2009

- [51]<http://ekiga.org/> En ligne. Consulté le 25 avril 2009
- [52]<http://www.brothersoft.com> En ligne. Consulté le 20 janvier 2009
- [53]<http://www.zoiper.com/> En ligne. Consulté le 14 mars 2009
- [54]<http://www.iptel.org/ser/> En ligne. Consulté le 17 mars 2009
- [55]<http://www.rtpproxy.org/> En ligne. Consulté le 20 aout 2008
- [56]<http://yate.null.ro/pmwiki/> En ligne. Consulté le 17 juillet 2008