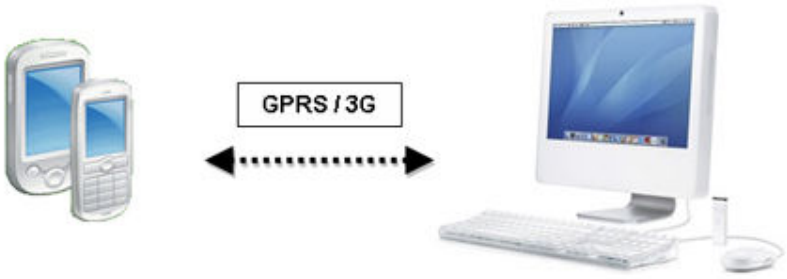
INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa



Controlo remoto de um PC através de uma plataforma 3G em comutação de pacotes

52208 - Nuno Filipe Canoilas Freire

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente:	Prof. José António Beltran Gerald
Orientador:	Prof. Mário Serafim Nunes
Vogais:	Prof. Rui Gustavo Crespo

Novembro de 2007

Agradecimentos

No decorrer desta dissertação de mestrado, muitos foram os incentivos e apoios que me foram dirigidos. Quero agradecer, em primeiro lugar, ao Professor Orientador, Mário Serafim Nunes, e aos acompanhantes do trabalho realizado, Luís Silva, Carlos Anjos e Nelson Escravana, sem os quais, certamente não estaria concluído. Gostaria também de agradecer os preciosos comentários e sugestões realizadas pelos colegas de curso assim como todo o apoio oferecido pelos amigos e família.

Resumo

O desenvolvimento de novos serviços de dados suportados por novas redes de telecomunicações, veio permitir a implementação de determinadas aplicações sobre comunicações móveis que antigamente apenas faziam sentido em computadores pessoais com acesso à Internet. Deste modo, o objectivo deste projecto foi o de desenvolver uma aplicação que permitisse o controlo remoto de um PC, através do uso de um simples terminal móvel. Um telemóvel não tem uma interface apropriada para controlar remotamente um PC, no entanto em situações particulares, pode-se tornar numa ferramenta bastante útil.

Tendo em consideração as limitações impostas pelos dispositivos móveis e pelas redes de dados que os suportam, foram realizados estudos de viabilidade do projecto, contornaram-se obstáculos introduzidos pelas API's utilizadas e teve-se especial atenção para limitar o mínimo possível, em termos de plataformas alvo, o uso da aplicação resultante. Para isso foram disponibilizados vários modos de funcionamento, que se adequam a telemóveis com diferentes capacidades de memória e de processamento.

Algumas das principais inovações introduzidas passam por permitir a transferência de som do computador remoto para o telemóvel, concedendo ao utilizador uma ideia mais aproximada de estar efectivamente sentado em frente ao computador a ser controlado, e passam também por permitir definir perfis de utilizador, restringindo o acesso dos terminais móveis a aplicações especificas e introduzindo restrições de segurança.

Por último, para estudo das capacidades e características das redes móveis celulares, foram efectuados testes com o intuito de se estimarem alguns parâmetros específicos destas, nomeadamente a latência e o tempo de ida e volta.

Palavras-chave

Controlo remoto de computadores; Redes móveis celulares; Telemóveis; Interfaces de entrada e saída; Protocolo de comunicação; Java

Abstract

The development of new data services supported by new telecommunication networks, allowed the conception of a variety of software applications over mobile communications that usually only made sense in personal computers with Internet access. Knowing this, the purpose of this work was to develop an application that allows the remote control of a desktop computer using a mobile device. Although a mobile phone is not the friendliest interface to remote control a desktop computer, it can however, become an important tool in several situations.

Considering all the limitations imposed by the mobile devices and by the data network that supports them, several tests were made to determine the viability of the project and some work was done to solve the problems related with the APIs used in this project. Also, through the development of several navigation and scaling modes, special attention was given in order to make the application work with as many different types of mobile devices as possible, both in terms of memory and processing capabilities.

Original features, like the possibility of sound transfer from the remote computer to the mobile device, giving the user a closest idea of really being sited in front of the computer, and the definition of user profiles, making the access to a specific application much easier by the mobile device, were introduced in this work.

Finally, tests were made to measure specific parameters of the mobile cellular networks services used (i.e. throughput and round trip time).

Key Words

PC remote control; Mobile phones; Mobile cellular networks; I/O interface; Communication protocol; Java

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

3G	Terceira Geração (terminais móveis)
AMC	Adaptive Modulation and Coding
API	Application Programming interface
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
DES	Data Encryption Standard
DoS	Denial of Service
FDD	Frequency Division Duplex
FDMA	Frequency Division Multiple Access
GPL	GNU General Public License
GPRS	General Packet Radio Services
GSM	Global System for Mobile Communications
HSDPA	High-Speed Downlink Packet Access
HTTP	Hyper-Text Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
J2ME	Java 2 Micro Edition
JPEG	Joint Photographic Experts Group
MMAPI	Mobile Media API
MIDP	Mobile Information Device Profile
MTU	Maximum Transmission Unit
PCM	Pulse Code Modulation
QoS	Quality of Service
RFB	Remote Frame Buffer
RRE	Rise-and-Run-length Encoding
SSH	Secure Shell
TCP/IP	Transfer Control Protocol / Internet Protocol
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
UMTS	Universal Mobile Telecommunications System
VNC	Virtual Network Computing
VPN	Virtual Private Network
RMS	Record Management System
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
ITU	International Telecommunication Union
W-CDMA	Wideband Code Division Multiple Access

Capítulo I. Introdução

I.1 Introdução

As diferenças de funcionalidade existentes entre telemóveis e computadores pessoais são cada vez menores, mas ainda assim existem devido aos requisitos específicos dos terminais móveis. Com a crescente evolução das capacidades dos telemóveis e das respectivas redes, estas poderão servir de ponte para aproximar, cada vez mais, um vulgar telemóvel de um computador pessoal. O objectivo deste trabalho foi o de desenvolver uma plataforma que permitisse aceder e controlar um computador através de um terminal móvel, recorrendo para isso a uma ligação de dados GPRS ou UMTS (3G). Deve-se no entanto ter em consideração as limitações dos telemóveis comuns, em termos de dimensões do ecrã, memória e interface de entrada.

É importante diferenciar entre ferramentas de acesso remoto e ferramentas de controlo remoto do PC [6]**:** nas primeiras, as aplicações existem no cliente e o servidor de acesso remoto é basicamente um comutador multi-protocolo baseado em *software*, dedicado a facilitar a comunicação entre os clientes de acesso remoto e os recursos na rede em que está inserido; as segundas foram concebidas para controlar remotamente o computador alvo, partilhando o ecrã, teclado e rato, através de uma ligação à distância, sendo as aplicações concretizadas no servidor.

Deste modo, a ideia consiste em, desenvolver uma ferramenta para controlo remoto de PC's, permitindo a um telemóvel (cliente), aceder facilmente a um computador pessoal que se encontre ligado à Internet (servidor), com o intuito de o controlar, visualizar todo o ambiente de trabalho e escutar o áudio que estiver a ser reproduzido no mesmo. Para este efeito, resolveu-se utilizar uma arquitectura cliente – servidor, pela sua simplicidade e utilidade prática e por não requerer a utilização de mais componentes alheios ao utilizador.

Optou-se por utilizar no servidor (computador a ser controlado) uma aplicação de distribuição gratuita, *open-source* e multi-plataforma, de modo a limitar o mínimo possível, em termos de plataformas alvo, o uso da aplicação a ser desenvolvida. Para este efeito, foi escolhido o *software* VNC (*Virtual Network Computing*) [1], que é um sistema de cliente "ultra-leve", largamente difundido, baseado num protocolo simples e independente da plataforma que permite, através da interface remota, visualizar e controlar o ambiente de trabalho do computador onde é instalado. O seu desenho assume o mínimo de requisitos acerca do cliente, facilitando o desenvolvimento de clientes nos mais variados tipos de *hardware* (no caso deste projecto, no *hardware* limitado dos telemóveis). Possui ainda características que permitem reduzir a largura de banda utilizada, como por exemplo, reduzindo o número de cores a utilizar, retirando o *Wallpaper* e padrões de fundo, permitindo transferir só as partes que sofrem alterações no ambiente de trabalho, fazendo actualizações só quando solicitadas pelo cliente, etc. Estas características tornam-se bastante úteis dadas as capacidades limitadas das redes móveis celulares e o preço cobrado pelas operadoras ao *kilobyte* transferido. Esta aplicação permite ainda que vários utilizadores estejam ligados ao mesmo servidor ao mesmo tempo, possibilitando o desenvolvimento de trabalho cooperativo.

No lado do cliente (terminal móvel), resolveu-se utilizar *Java* (J2ME), no seu formato MIDP1.0/CLDC1.0, pois a maioria dos telemóveis existentes no mercado suportam o uso desta tecnologia, tentando-se assim abranger o máximo de plataformas móveis possíveis. Existem já alguns programas disponíveis, concebidos para este efeito, sendo um dos objectivos deste projecto, o de melhorar e contribuir para a aplicação *open-source* existente (http://j2mevnc.sourceforge.net/) e adicionar-lhe novas funcionalidades. Nomeadamente, acrescentou-se a possibilidade de transferência de áudio entre o servidor e o cliente (**Secção IV.3**), tendo em consideração que a maior parte dos telemóveis actuais não suportam *streaming* sobre a API do Java utilizada (MMAPI [12]) mas apenas em *players* criados na linguagem nativa aos mesmos. Permitiu-se a definição de perfis de utilizador (**Secção IV.4**), ou seja, quando um utilizador se liga, dependendo do que estiver definido no servidor para ele, este poderá aceder a uma aplicação específica em vez de ter acesso a todo o ambiente de trabalho. Acrescentou-se a possibilidade de as alterações ao factor de escala da imagem visível no ecrã do telemóvel, serem efectuadas no lado do servidor (**Secção IV.2**) para minimizar os requisitos de processamento exigidos aos dispositivos móveis. Por último, tornou-se o uso da aplicação mais prático, directo e parecido com o existente nos computadores, tendo sempre em consideração as limitações impostas pelas interfaces dos telemóveis (**Secção IV.1** e **IV.2**).

Para a comunicação entre telemóvel e computador pessoal (cliente e servidor), utiliza-se no telemóvel as actuais redes móveis celulares existentes (GPRS e UMTS) e utiliza-se qualquer tipo de acesso à Internet por parte do computador. Para provar a validade deste projecto que opera sobre redes sem fios com características particulares, foi ainda realizado o estudo do funcionamento destas redes, dos seus parâmetros e das suas limitações (**Secção V.5**).

I.2 Enquadramento

Para o desenvolvimento deste projecto, começou-se por analisar as limitações das redes móveis celulares, para deduzir se seriam capazes de suportar os requisitos do projecto. Verificou-se que as redes UMTS (3G) têm uma capacidade máxima, dependente do contracto com a operadora, entre 384 kbps e 14 Mbps teóricos (utilizando HSDPA [3]). Como se pretende abranger o mais vasto leque de telemóveis possível, verificou-se também que as redes GPRS permitem entre 28 kbps e 64 kbps de *Downlink* e 14 kbps de *Uplink*, valores estes bastante mais limitativos.

Desde modo, tendo em consideração as capacidades e limitações das redes, analisaram-se as várias alternativas possíveis para transferir o ambiente de trabalho do servidor para o cliente, para assim optar por uma via de trabalho. Algumas das hipóteses ponderadas foram:

- **Sequência de imagens *JPEG*.** Uma imagem de 176x208 *pixeis* (dimensões do ecrã de um telemóvel Nokia 6600) fica codificada neste formato com cerca de 10 kB. Deste modo, podia-se reduzir as dimensões do ambiente de trabalho do computador para as dimensões do ecrã do telemóvel durante a compressão da imagem e enviá-la, correndo o risco da imagem ser imperceptível por estar demasiado pequena. Em alternativa, poder-se-ia transferir o ambiente de trabalho em secções com as dimensões do ecrã do telemóvel, “cortados” em redor da

posição do cursor do rato. Desde que o ritmo de actualizações não fosse bastante elevado esta hipótese pareceu-nos factível.

- **Compressão vídeo em mpeg-4.** A ideia seria a de realizar *streaming* do ambiente de trabalho do servidor, para o telemóvel. No entanto, foram encontradas algumas contrariedades, que numa primeira abordagem, não tornaram esta opção apelativa. Existiam duas soluções possíveis:
  - Utilizar o *player* nativo do telemóvel para aceder ao servidor e receber em *streaming* o ambiente de trabalho remoto. Tornava-se depois necessário conceber um método para interacção com o servidor, pois os *players* comuns apenas permitem receber informação.
  - Criar uma aplicação em J2ME para receber o *streaming* enviado pelo servidor e desenvolver um protocolo de comunicação para se poder interagir com este. Esta opção não se revelou exequível pelas dificuldades encontradas na grande maioria dos telemóveis em realizar *streaming* RTSP utilizando o J2ME.

- **VNC *vs* Criar uma nova aplicação de servidor.** No caso de se criar uma aplicação de raiz para correr no computador, esta deveria ser multi-plataforma, para limitar o mínimo possível a sua utilização. A alternativa seria utilizar a aplicação VNC que é de distribuição gratuita e já é independente da plataforma utilizada. O servidor de VNC tem também várias possibilidades para reduzir a largura de banda utilizada, tais como, retirar o papel de fundo do ambiente de trabalho (*Wallpaper*), reduzir o número de cores utilizadas, vários tipos de codificações para os formatos dos *pixeis* enviados, permite enviar apenas as secções do *desktop* que sofrem alterações, etc. Comparando com outras soluções existentes no mesmo âmbito, resolveu-se optar pelo VNC, por não ser uma solução proprietária e por disponibilizar o código sobre licença GPL [24]. Tudo isto fez do VNC, numa primeira aproximação, o candidato ideal para correr no computador alvo.
  Para utilizar o servidor VNC será necessário construir um cliente para correr no telemóvel, compatível com o protocolo RFB (*Remote Frame Buffer*) [2] utilizado por este.

- **Novo cliente *vs* Cliente *open-source* existente.** Tendo-se descoberto que já existem algumas aplicações concebidas neste âmbito, e sendo que a ideia seria a de desenvolver a aplicação em *Java* para ser compatível com o maior número possível de terminais móveis, optou-se por contribuir para um projecto *open-source* existente, que está desenvolvido em *Java* e que utiliza o protocolo RFB, descrito no **Anexo A**. Como este cliente está numa fase já funcional mas ainda bastante limitada (o seu desenvolvimento está parado desde 2005), revelou-se conveniente contribuir para este projecto, continuando o seu desenvolvimento.

Analisadas as hipóteses descritas anteriormente, chegou-se à conclusão que a melhor opção a adoptar para o desenvolvimento deste projecto, seria a de utilizar como base o servidor VNC, assim

como o cliente VNC existente para telemóveis, desenvolvendo todas as funcionalidades adicionais que se considerem necessárias para cumprir os requisitos identificados. A alternativa de construir tudo de raiz levaria mais tempo e implicaria um maior risco, alcançando-se no fim, provavelmente, um desenvolvimento análogo ao existente.

Deste modo, todo o projecto baseia-se na aplicação VNC, que é um sistema de partilha do ambiente gráfico de um computador, que utiliza o protocolo RFB para controlo remoto deste. Para alcançar tal objectivo, transmitem-se os eventos relacionados com o teclado e com o rato de um dispositivo para o outro, sendo as actualizações ao ambiente gráfico transferidas pela rede no sentido inverso. O VNC é independente da plataforma que o suporta. Deste modo, um cliente de VNC que esteja a operar sobre um qualquer sistema operativo, pode-se ligar a um servidor de VNC a funcionar sobre outro qualquer sistema operativo, existindo clientes e servidores desenvolvidos para quase todos os sistemas com ambientes gráficos conhecidos e também para *Java*. A utilização mais popular desta tecnologia inclui a assistência remota e o acesso a ficheiros no computador do trabalho a partir do computador de casa ou vice-versa. O VNC foi desenvolvido pela AT&T e o código fonte original e muitos dos seus derivados são *open-source* sobre licença GPL. O sistema VNC consiste num cliente, num servidor e num protocolo de comunicação (ver **Figura I.1**):

- O servidor de VNC é o programa que opera na máquina que partilha o seu ambiente de trabalho.
- O cliente de VNC é o programa que observa e interage com o servidor.
- O protocolo de VNC é bastante simples e baseia-se numa única primitiva gráfica: “Colocar um rectângulo de *pixeis* na posição (X, Y) especificada”.

O servidor envia rectângulos de dimensões reduzidas para o cliente, correspondentes a porções do seu ambiente de trabalho. Sendo que na sua forma mais simples o protocolo de VNC pode ocupar muita largura de banda, vários métodos foram criados para reduzir o *overhead* da comunicação. Por exemplo, existem vários tipos de codificações dos dados, podendo a escolha do tipo a utilizar ser negociado entre o cliente e o servidor no início de uma sessão. O tipo de codificação mais simples e que é suportado por todos os clientes e servidores é a codificação *Raw* (**Secção A.5.5.1**), onde todos os *pixeis* são enviados sem serem codificados, num varrimento em linha da esquerda para a direita. Depois de todo o ambiente de trabalho original ser transferido, apenas as porções deste que sofrerem alterações são de novo enviadas para o cliente. Esta codificação funciona bem se apenas pequenas porções do ambiente de trabalho sofrerem alterações, como sendo o ponteiro do rato a deslocar-se ou algo a ser escrito no ecrã, no entanto, a exigência de largura de banda torna-se bastante elevada se um número considerável de *pixeis* sofrerem alterações ao mesmo tempo, como sendo quando se está a visualizar um vídeo em modo ecrã completo.

Por omissão, o VNC não é um protocolo seguro. Apesar das palavras-chave não serem enviadas de forma legível para o utilizador comum, tentativas de força bruta para as descobrir podem ser bem sucedidas se tanto a cifra como a palavra-chave cifrada forem apanhadas na rede. Por esta razão recomenda-se que as palavras-chave utilizadas sejam constituídas por pelo menos 8 caracteres. No entanto, o VNC pode funcionar sobre uma sessão SSH ou VPN, o que adiciona uma camada de

segurança extra, a toda a sessão. Ambos os mecanismos referidos estão disponíveis para os principais sistemas operativos.

No VNC, os servidores fornecem não apenas dados e aplicações, mas também o ambiente de trabalho completo que pode ser acedido de qualquer máquina ligada à Internet. Para além disso, o VNC permite que um único ambiente de trabalho seja acedido de vários locais simultaneamente, suportando a partilha de aplicações no âmbito de trabalhos cooperativos. Ou seja, pode oferecer suporte computacional aos indivíduos que tentam resolver um problema em colaboração com outros, sem que todos estejam no mesmo local, ao mesmo tempo. Os sistemas cooperativos permitem a comunicação de ideias, a partilha de recursos e a coordenação dos esforços de trabalho. A sua meta é permitir o trabalho em conjunto de forma simples e eficaz, ajudando a comunicar, coordenar e colaborar. Um exemplo que pode ser referido é o de desenvolvimento de aplicações em equipa. Se os membros da equipa não se encontrarem na proximidade uns dos outros, torna-se necessário recorrer a métodos alternativos de comunicação para debaterem ideias, tirarem dúvidas, fundamentarem opções de projecto, delinear tarefas individuais, etc. Neste caso, toda a equipa pode combinar aceder por VNC ao computador de um dos seus membros e este poderá explicar através de uma pequena demonstração o que têm vindo a fazer, pedir ajuda numa secção de código que apresenta um erro, ou inclusive, utilizando uma ferramenta de desenho, tentar explicar as opções de projecto consideradas ou tentar ilustrar algum conceito. Durante todo este processo os restantes membros da equipa podem não só visualizar o que se está a passar no computador remoto mas também interagir com o mesmo.

O protocolo RFB utilizado, não mantém no cliente informação de estado. Deste modo, se um cliente se desligar e se voltar a ligar ao mesmo servidor, o estado do ambiente de trabalho no servidor é preservado. Se um segundo cliente se conectar ao mesmo servidor, ele irá ter acesso exactamente ao mesmo ambiente de trabalho que o primeiro cliente e que qualquer outro cliente que se ligue posteriormente. Com efeito, obtém-se assim o que pode ser chamado de “ambiente de trabalho móvel”, ou seja, onde quer que exista uma ligação à Internet, o utilizador pode aceder às suas aplicações pessoais e o estado destas é preservado entre acessos.

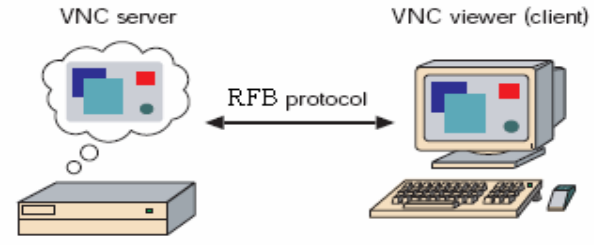


Figura I.1 – Arquitectura da VNC.

I.3 Estrutura do documento

- Primeiro capítulo – É realizada uma pequena introdução ao trabalho desenvolvido e é apresentado um resumo do raciocínio efectuado durante a fase de projecto do sistema, onde foi estabelecido a base para a sua implementação. Foi então realizado um enquadramento à abordagem seleccionada.
- Segundo capítulo – Apresenta um pequeno enquadramento histórico das redes móveis celulares e uma descrição geral das características das mesmas.
- Terceiro capítulo – É realizada uma análise global às soluções existentes e é efectuada a escolha da aplicação de cliente e de servidor a utilizar.
- Quarto capítulo – São descritas ao pormenor todas as alterações efectuadas no cliente e no servidor, revelando as novas funcionalidades implementadas. Documentou-se todas as decisões de projecto e problemas encontrados.
- Quinto capítulo – São descritos os vários métodos implementados para aquisição de dados, que permitem efectuar uma análise às redes móveis celulares e aos tempos inerentes ao bom funcionamento da aplicação. São ainda apresentados os resultados de alguns testes realizados.
- Sexto capítulo – Apresentam-se as conclusões e perspectivas de trabalho futuro.
- Os Anexos contêm informação adicional:
  - Anexo A – Apresentação do protocolo de comunicação RFB utilizado pelo VNC.
  - Anexo B – Manual do Utilizador do cliente e das novas funcionalidades do servidor, que foi disponibilizado à comunidade *open-source* onde o projecto original se inclui.
  - Anexo C – São descritos os algoritmos (pseudo-código) de compressão e descompressão dos pacotes de áudio.
  - Anexo D – É apresentado um exemplo de um ficheiro onde se podem analisar as repetições de símbolos ocorridos numa amostra de áudio.

Capítulo II. Redes Móveis Celulares

O sistema de controlo remoto de um computador pessoal proposto neste projecto, pode ser suportado por várias redes de acesso móvel, tais como o GSM, GPRS, UMTS e HSDPA, permitindo deste modo, utilizar as várias gerações disponíveis até hoje. Neste capítulo são abordadas as características gerais de cada uma destas tecnologias de acesso móvel, dando-se particular ênfase ao ritmo máximo de transmissão disponibilizado para *downlink*, sendo que esta é a principal característica necessária para a viabilidade do trabalho em questão.

Os serviços disponibilizados pelas redes móveis celulares podem ser divididos em três categorias:

- Serviços de Suporte ou de Transporte – Permitem efectuar o transporte de dados entre terminais móveis.
- Tele-Serviços – Definidos em função dos serviços de transporte e disponibilizam as capacidades necessárias, incluindo funções no terminal, que permitem a comunicação entre utilizadores.
- Serviços Suplementares – Serviços adicionais oferecidos pelos operadores, tais como por exemplo, o envio ou supressão da identificação do chamador, possibilidade de chamadas em conferência, chamada em espera e reencaminhamento de chamadas.

Dependendo da rede utilizada, os serviços são estabelecidos por comutação de circuitos ou de pacotes, sendo que a principal diferença e vantagem dos serviços que utilizam a comutação de pacotes sobre os serviços com estabelecimento de circuitos, prende-se com o facto de não utilizarem os recursos da rede quando não se está a enviar ou a receber informação, podendo ser estabelecida uma ligação permanente, que permita a transmissão de dados de uma forma contínua. Uma outra vantagem dos serviços estabelecidos por comutação de pacotes é a facturação ser efectuada sobre o volume de informação transferida e não sobre o tempo da ligação. Ao contrário do que acontece numa ligação através da comutação de circuitos, os recursos atribuídos numa ligação em comutação de pacotes não são constantes, o que faz com que a qualidade de serviço também não seja constante.

II.1 GSM

O GSM é uma norma de comunicações móveis, adoptada inicialmente ao nível da Europa a partir do ano de 1992, para fornecimento de serviços de comunicação baseados em comunicação digital. Concretamente em Portugal esta tecnologia surgiu inicialmente através do operador TMN e em seguida pelos operadores Telecel e Optimus. Com o intuito de substituir os sistemas analógicos utilizados até então na Europa, o GSM foi concebido para dar suporte principalmente a serviços de voz mas posteriormente passou a suportar também serviços de dados. O GSM 1800 foi disponibilizado também a nível Europeu no ano de 1997 com a finalidade de diminuir a sobrecarga em termos de capacidade do GSM 900. No que diz respeito ao espectro de frequência, o GSM 900 utiliza a banda entre 935 e 960 MHz nos canais de *downlink* e utiliza a banda entre 890 e 915 MHz

para os canais de *uplink*, enquanto que o GSM 1800, utiliza a bandas entre 1805 e 1880 MHz para os canais de *downlink* e utiliza a banda entre 1710 e 1785 MHz para os canais de *uplink*. No GSM 900 cada banda é subdividida em 124 portadoras e no GSM 1800 em 374 portadoras, espaçadas de 200 kHz. Cada portadora é ainda dividida em 8 *time slots*, correspondendo a cada *time slot* um canal a atribuir a cada utilizador. É uma tecnologia que utiliza para acesso ao canal rádio a combinação das técnicas de divisão no tempo (TDMA) e na frequência (FDMA), dadas as limitações do espectro.

O GSM foi desenvolvido tendo como principal objectivo a conquista do mercado de voz e apenas posteriormente surgiram alguns serviços de dados limitados pelo baixo ritmo de transmissão disponível. Tais serviços apresentam um ritmo de transmissão máxima de 14,4 kbps e uma vez que esta tecnologia utiliza a comutação de circuitos, é evidente a desvantagem na tarifa aplicada a um serviço de dados como o acesso à Internet, isto é, a tarifa é aplicada durante o tempo em que o circuito esteja estabelecido e não sobre a quantidade de informação transmitida.

II.2 GPRS

O crescimento das aplicações de dados como o acesso à Internet, *e-mail* e entretenimento, levou à necessidade de desenvolver soluções que permitissem a transmissão de dados a ritmos superiores em comutação de pacotes. Em 1999 foi lançado em Portugal a nível comercial também pelos operadores TMN, Vodafone (antiga Telecel), e Optimus, a tecnologia GPRS [14] que está inserida na segunda geração e meia (2.5G), e foi a principal tecnologia que serviu como ponte entre 2G e 3G. O GSM foi implementado tendo como principal objectivo o serviço de voz, enquanto que o GPRS utilizando a rede GSM, pretende disponibilizar serviços de transmissão de dados, como por exemplo o acesso à Internet. Com este propósito, o GPRS utiliza múltiplos *time slots* para efectuar a transmissão dos pacotes de dados, com a diferença que não existe uma reserva permanente dos mesmos. Os *time slots* são reservados conforme a necessidade de transmitir informação, conseguindo-se desta forma um serviço de dados com ligação permanente (*always on*), sem a necessidade de reservar continuamente os *time slots* para o transporte de dados. Deste modo, uma das vantagens é a facturação ser efectuada sobre o volume de informação transaccionado e não sobre o tempo da ligação.

No GSM por cada chamada de voz é atribuído um *time slot*. No GPRS, de modo a maximizar o ritmo de transmissão podem ser atribuídos ao mesmo terminal vários *time slots*. As especificações definem 29 classes para os terminais, conforme o número de *time slots* utilizados na recepção ou transmissão. O débito obtido está directamente relacionado com três factores: esquema de codificação utilizado no canal, número de *time slots* suportados pelo dispositivo móvel e número de utilizadores de GSM e GPRS na célula. Os esquemas de codificação permitem a detecção e correcção de erros em caso de perda de dados e são utilizados na interface rádio dependendo da qualidade da ligação. São definidos 4 esquemas de codificação para o GPRS, tal como se apresenta na **Tabela II.1**.

Tabela II.1 – Esquemas de codificação do GPRS.

Codificação	Ritmo máximo de dados [kbit/s] (Camada LLC)	Ritmo máximo de dados [kbit/s] Canais físicos	C/I min[dB]
CS-1	8	9,05	-6
CS-2	12	12,4	-9
CS-3	14	15,6	-12
CS-4	20	21,4	-17

Pode-se verificar que quanto melhor for a cobertura da célula, ou seja, quanto maior for a relação sinal ruído (C/I), menos robusto deve ser o esquema de codificação a utilizar. Sendo o débito obtido em função do esquema de codificação e do número de *time slots*, os débitos máximos em kbps são apresentados na **Tabela II.2**. Com esta tecnologia é possível atingir um ritmo máximo teórico de 171,2 kbps para o esquema com menos redundância (CS-4) e com a utilização dos 8 *time slots*.

O número de *time slots* utilizado depende da classe do terminal móvel e da configuração da rede. Na configuração da rede GPRS, o operador pode reservar um determinado número de *time slots* só para tráfego de dados, ou então, pode dar prioridade à voz e o tráfego de dados utiliza apenas os canais livres. Desta forma o tráfego de dados depende do número de utilizadores activos de GSM numa célula.

Na realidade os operadores utilizam normalmente o esquema de codificação CS-2 e a maioria dos terminais móveis são de classe *multislot* 6 (*Slots* - Rx:2 e Tx:2 ou Rx:3 e Tx:1), o que implica um débito em *uplink* entre 13,4 e 26,8 kbps e em *downlink* entre 26,8 e 40,2 kbps. Estes débitos estão ainda dependentes do nível de congestionamento da célula que serve o terminal ou da existência de *time slots* disponíveis.

Tabela II.2 - Débito máximo em função do esquema de codificação e do número de *time slots.*

Slots	CS-1 [kbit/s]	CS-2 [kbit/s]	CS-3 [kbit/s]	CS-4 [kbit/s]
1	9,05	13,4	15,6	21,4
2	18,1	26,8	31,2	42,8
3	27,17	40,2	46,8	64,2
4	36,2	53,6	62,4	85,6
5	45,25	67,0	78,0	107,0
6	54,3	80,4	93,6	128,4
7	63,35	93,8	109,2	149,9
8	72,4	107,2	124,8	171,2

II.3 UMTS

UMTS [15] designada por "*Universal Mobile Telecommunication System*" é a sigla utilizada para especificar o padrão 3G (terceira geração), sendo esta a evolução para as operadoras de GSM. Apesar das principais especificações desta tecnologia terem sido definidas entre o ano de 2000 e 2002, a sua introdução em Portugal foi efectuada pelo operador Vodafone no princípio do ano 2004

logo seguida da TMN em Abril do mesmo ano e por último a Optimus. A necessidade de desenvolver esta nova tecnologia, prendeu-se com o facto de haver um grande aumento de utilizadores de comunicações móveis e também porque o sistema GSM começou a ficar saturado no que diz respeito às frequências de rádio que lhe foram destinadas. O desenvolvimento de novas aplicações e a melhoria da qualidade de serviço de algumas aplicações já existentes no 2.5G foi outra das razões para implementação da 3G.

Ao nível de frequências o UMTS utiliza a faixa entre 1900 e 2200 MHz, onde 155 MHz são para a componente terrestre e 60 MHz são para a componente de satélite. As bandas entre 1900 e 1920 MHz e entre 2010 e 2025 MHz são utilizadas pelo modo TDD (*Time Division Duplex*), ambas para *uplink* e *downlink*, as bandas entre 1920 e 1980 MHz e entre 2110 e 2170 MHz são utilizadas pelo modo FDD (*Frequency Division Duplex*), em *uplink* e *downlink* respectivamente e as bandas entre 1980 e 2010 MHz e entre 2170 e 2200 MHz são utilizadas pelo modo MSS (*Mobile Satellite System*), em *uplink* e *downlink* correspondentemente.

A interface rádio utilizada no UMTS é designada por WCDMA (*Wide Code Division Multiple Access*). Esta sigla provém do facto de se tratar de um sistema de banda larga com base na tecnologia CDMA (*Code Division Multiple Access*) que efectua um espalhamento do espectro de um determinado sinal na frequência, ou seja, transforma um sinal de banda estreita num sinal de banda larga. A cada utilizador é atribuído um código através do qual é efectuado o espalhamento do sinal quando o utilizador pretende transmitir. Na recepção, através do conhecimento do mesmo código é efectuado o processo inverso. Como os códigos são ortogonais entre si é possível ter mais que um utilizador a partilhar a mesma frequência sem ocorrer o risco de perda de informação.

Os ritmos de transmissão estão directamente relacionadas com o tipo de célula (área de cobertura), e com a mobilidade do terminal móvel, sendo que em situações ideais, ou seja, zonas próximas da estação base e com pouca mobilidade, o débito pode atingir os 2 Mbps. Pelo contrário, em zonas mais afastadas da estação de base e para mobilidades superiores o débito pode atingir valores de 144 kbps. A ITU (International Telecommunication Union) definiu três limites de débito associados à mobilidade:

- Alta: Ritmo de transmissão na ordem dos 144 kbps quando o utilizador estiver a viajar a mais de 120 km/h no exterior em ambientes rurais.
- Total: Ritmo de transmissão na ordem dos 384 kbps para peões que se estejam a deslocar a menos de 120 km/h no exterior, em ambientes citadinos.
- Limitada: Ritmo de transmissão na ordem dos 2Mbps para um utilizador que se esteja a deslocar a menos de 10 km/h, dentro de um edifício.

Actualmente em Portugal apenas o modo FDD está em funcionamento. No caso em que estejam ambos os modos a operar em conjunto, o modo FDD pode ser aplicado em ambientes macro e microcelulares, permitindo ligações simétricas e para elevada mobilidade, disponibilizando ritmos de transmissão até aos 384 kbps. Por sua vez o modo TDD pode ser utilizado em ambientes micro e picocelulares, permitindo ligações assimétricas e simétricas, disponibilizando ritmos de transmissão até 2 Mbps para baixa mobilidade.

II.4 HSDPA

O serviço HSDPA (*High Speed Downlink Packet Access*) [16] foi desenvolvido com base no WCDMA e tem como principais objectivos aumentar o ritmo de transmissão em *downlink* fornecido ao utilizador e melhorar a qualidade de serviço (QoS). O HSDPA atribui ao utilizador um canal de dados em *downlink* com um ritmo de transmissão máximo que pode chegar aos 14 Mbps. A utilização desta tecnologia é adequada para serviços em que o tráfego recebido é superior ao tráfego enviado, tal como acontece quando se navega na Internet ou na transmissão de vídeo em tempo real (*multicast*). Uma das principais técnicas empregues nesta tecnologia para obter um elevado ritmo de transmissão em *downlink* é a utilização de uma modulação e codificação adaptativa (AMC – Adaptive Modulation and Coding).

Na **Tabela II.3** é visível a evolução da tecnologia GSM, dando-se especial atenção aos ritmos de transmissão em *downlink* disponibilizados pelas várias gerações

Tabela II.3 - Evolução da Tecnologia GSM.

Geração	**2 G**	**2,5 G**	**3 G**	
Tecnologia	GSM	GPRS	WCDMA (UMTS)	WCDMA/HSDPA (UMTS)
Taxa de dados máxima teórica (kbit/s)	14,4	171,2	2.000	14.000
Taxa de dados média (kbit/s)	-	30-40	220-320	550-1100
Frequência de funcionamento (MHz)	900 e 1800	900 e 1800	2000	2000
Espaçamento entre portadoras (kHz)	200	200	5000	5000

Capítulo III. Análise de Soluções Existentes

A possibilidade de aceder e controlar remotamente um PC a partir de outro computador, há muito que é real. Inclusive, o número de produtos e ferramentas que permitem tal ocorrência aumentou, recentemente, em número e capacidades. No entanto, estes produtos são ainda em escasso número quando toca a aplicações para correrem sobre plataformas móveis, dadas as suas limitações inerentes.

Neste âmbito, a *SHAPE services* (http://www.shapeservices.com/) oferece algumas ferramentas de controlo remoto para correr sobre telemóveis ou PDA's, todas elas proprietárias e comerciais, não sendo este facto apreciado em meios académicos. Entre elas incluem-se três variantes: RDM+™ (*Remote Desktop for Mobiles*), TSMobiles™ (*Terminal Service for Mobiles*) e VNC+ (*Virtual Network Computing for Mobiles*).

A primeira solução, RDM+, é um serviço composto por três componentes (ver **Figura III.1**), que permite aceder ao computador de casa ou da empresa, quando não se pode abrir um acesso directo a este por alguma razão. O primeiro destes componentes é uma aplicação a ser instalado no computador alvo (serviço local), que serve para registar e autenticar o computador no segundo componente denominado *RDM Online Service*. Este serviço local inicia uma comunicação com o servidor RDM fazendo um pedido HTTP (RFC 1945/2616) para verificar se existem novas ligações pendentes. Por último, o cliente móvel (o último componente) é instalado no telemóvel. Este mecanismo providência um acesso seguro ao computador remoto, não sendo necessário abrir "buracos" na *firewall* da empresa ou proceder a configurações especiais nos comutadores (*routers*). O RDM+ é iniciado no telemóvel e depois de se introduzir o nome de utilizador e palavra-chave e de se seleccionar *Connect*, a aplicação envia um pedido cifrado ao *RDM Online Service*. Este por sua vez escuta o pedido de ligação e caso o computador alvo se encontre registado, é então estabelecida uma sessão entre o cliente e o PC.

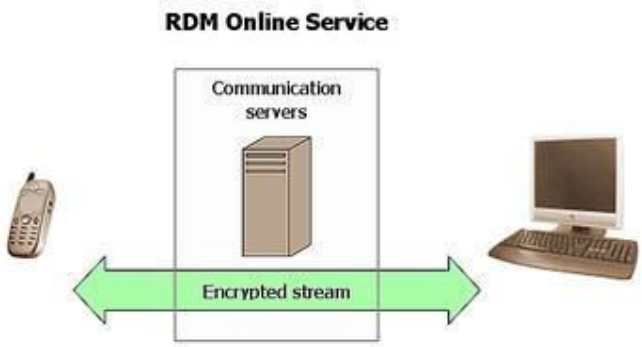


Figura III.1– Esquema do serviço RDM+.

A segunda solução, TSMobiles, é essencialmente um cliente móvel baseado em RDP (*Windows Remote Desktop Protocol*), que permite ter acesso a qualquer Sistema Operativo Windows através de

um telemóvel, bastando para tal que o computador alvo tenha o *Terminal Service* (Windows NT/2000/2003) ou o *Windows Remote Desktop* (Windows XP e Vista) instalado.

A terceira solução, VNC+, é um cliente móvel baseado em VNC que permite ter acesso a plataformas Windows, Macintosh ou Unix desde que contenham um servidor de VNC instalado (gratuito e *open-source*).

A **Tabela III.1** [4] apresenta um resumo das características de cada uma destas ferramentas, permitindo uma melhor comparação entre elas.

Tabela III.1 – Comparação entre as três soluções oferecidas pela *SHAPE services* [4].

Aplicação de controlo remoto	**RDM+**	**VNC+**	**TSMobiles**	
Sistema Operativo	Microsoft Windows NT / 2000 / XP / 2003	Microsoft Windows 98 / NT / 2000 / XP / 2003, Linux / Unix e MacOS	Microsoft Windows NT / 2000 / XP / 2003	Microsoft Windows XP Professional
Quantidade de utilizadores	Vários utilizadores podem aceder ao ambiente de trabalho remoto no mesmo instante e interagir com este.	Vários utilizadores podem aceder ao ambiente de trabalho remoto no mesmo instante e interagir com este.	Vários utilizadores podem aceder ao mesmo tempo ao computador remoto e terão acesso a um ambiente de trabalho virtual que não estará disponível para os outros utilizadores.	Apenas um utilizador poderá aceder ao ambiente de trabalho em cada instante.
Requerimentos de ligação	Não é necessário um acesso directo ao computador alvo, sendo que este pode estar por detrás de uma firewall / /proxy.	É necessário um acesso directo através da Internet ao porto 5900 no computador alvo. A ligação a este porto deve ser concedido pela firewall / proxy.	É necessário um acesso directo através da Internet ao porto 3389 no computador alvo. A ligação a este porto deve ser concedido pela firewall / proxy.	
Aplicações adicionais	Requer a instalação da aplicação adicional proprietária no computador alvo.	Requer a instalação do software grátis no computador alvo: UltraVNC RealVNC TightVNC	Não requer aplicações adicionais	
Tipo de ligação	TCP (directo) ou HTTP	TCP (directo)	TCP (directo)	
Troca de ficheiros	Sim	-	-	
Protocolo	RDM (Proprietário)	VNC/RFB	RDP (Windows Remote Desktop Protocol)	

Todas estas ferramentas requerem que os telemóveis suportem Java (J2ME) com a especificação MIDP2.0, que só se encontra disponível nos telemóveis mais recentes. Para não limitar o uso da aplicação resultante, neste projecto resolveu-se utilizar o MIDP1.0 por ser suportado por um maior número de plataformas móveis.

Outras duas aplicações disponíveis para este mesmo efeito são a VNC2Go (www.freeutils.net) e a j2mevnc (http://j2mevnc.sourceforge.net/), sendo que ambas são de distribuição gratuita, mas só a segunda é *open-source* com licença GPL. A primeira apresenta-se como um ferramenta mais simples, não permitindo alterações ao factor de escala do ambiente de trabalho e revelando-se mais limitada nas funcionalidades, comparando com as outras soluções. Tendo este facto em consideração, a escolha para cliente móvel recaiu sobre a segunda opção por ser *open-source*. Deste modo, não se despendeu tempo a conceber uma aplicação de raiz, visto que já existem soluções realizadas no mesmo âmbito.

Em termos de características e funcionalidades disponíveis nas várias aplicações mencionadas neste capítulo, pode ser visualizado um resumo das mais importantes na **Tabela III.2**.

Tabela III.2 – Resumo das principais características das várias aplicações de controlo remoto

Aplicação de controlo remoto	RDM+	TSMobile	VNC+	VNC2Go	J2MEVNC (versão original)	J2MEVNC (versão final)
Visualização em ecrã completo	Sim (requer MIDP2.0)	Sim (requer MIDP2.0)	Sim (requer MIDP2.0)	Não	Não	Não
Zoom	Sim	Sim	Sim	Não	Sim	Sim (melhorado)
Protecção por palavra-chave dos endereços armazenados (caso em que o telemóvel seja furtado ou perdido)	Sim	Sim	Sim	Não	Não	Sim
Orientação da imagem visível no ecrã do telemóvel	Sim (requer MIDP2.0)	Sim (requer MIDP2.0)	Sim (requer MIDP2.0)	Não	Não	Não
Livro de endereços	Sim	Sim	Sim	Não	Sim	Sim (melhorado)
Transferência de ficheiros	Sim	Não	Não	Não	Não	Não
Segurança durante a sessão (dados cifrados)	Sim	Sim	Não	Não	Não	Não
Permite associar algumas funcionalidades a teclas específicas do telemóvel	Sim	Sim	Não	Não	Não	Não
Clipboard	Sim	Sim	Sim	Não	Não	Sim
Disponibilização dos vários botões do rato e duplos cliques	Sim	Sim	Sim	Sim	Só clique simples e duplo-clique	Sim
Lista com combinações de teclas e teclas especiais	Sim (não exaustiva)	Sim (não exaustiva)	Sim (não exaustiva)	Sim (não exaustiva)	Apenas o ctrl, alt, shift, meta e o enter são disponibilizados e existe a possibilidade de o utilizador configurar combinações de teclas	Sim (não exaustiva)
Transferência de som	Não	Não	Não	Não	Não	Sim
Perfil de utilizadores	Não	Não	Não	Não	Não	Sim
Possibilidade de as alterações ao factor de escala serem efectuados pelo servidor	Não	Não	Não	Não	Não	Sim
Armazenamento em memória de todo o ambiente de trabalho possibilitando uma navegação sem necessidade de pedidos de actualização constantes	Não	Não	Não	Não	Não	Sim

Em termos de *software* necessário no servidor, a escolha recaiu no VNC por ser multi-plataforma, de distribuição gratuita e *open-source*, permitindo assim o desenvolvimento de novas funcionalidades que envolvam programação tanto no lado do cliente como no lado do servidor.

No caso do presente trabalho, foi mesmo necessário desenvolver em ambos os lados da arquitectura cliente – servidor, para permitir, por exemplo, a transferência de som, a realização das alterações ao factor de escala por parte do servidor e a definição de perfis de utilizador. Foi ainda

necessário estender o protocolo de comunicação utilizado para suportar tais funcionalidades. A escolha de entre as versões de servidores VNC disponíveis, recaiu sobre a versão "*WinVNC Version 3.3.3r7 with server-side scaling extensions*", pois mostrou-se mais acessível reproduzir nesta as funcionalidades existentes nas versões posteriores, do que reproduzir nas versões mais recentes a extensão que permite realizar as alterações ao factor de escala no lado do servidor. Funcionalidades como, o que fazer quando o último utilizador se desconecta, permitir retirar o *Wallpaper*, padrões de fundo e efeitos do Windows, entre outras que só estavam disponíveis em versões posteriores, foram implementadas nesta versão. Para isso utilizou-se a ideia existente e concretizou-se para o código da versão de VNC utilizada.

Capítulo IV. Novas Funcionalidades desenvolvidas

IV.1 Melhoramentos à aplicação existente

Tendo em consideração um dos objectivos deste trabalho que é a contribuição para o projecto *open-source* existente, foram adicionadas novas funcionalidades e realizados alguns melhoramentos à aplicação, tentando-se não alterar significativamente o modo de funcionamento desta. De seguida apresenta-se uma lista de novas funcionalidades e alterações efectuadas:

1. Quando se estabelecia uma conexão, apenas se recebia uma porção do ambiente de trabalho das dimensões do ecrã do telemóvel e o utilizador tinha de ir "navegando" para descobrir o resto do ambiente de trabalho.

   A aplicação passou a iniciar-se num modo em que todo o ambiente de trabalho encontra-se reduzido ao ponto de ser visível no ecrã do telemóvel (*zoom out*), sendo efectuado um pedido de actualização a todo o ambiente de trabalho mal se estabelece a ligação ao servidor. Deste modo, o utilizador obtém um panorama geral do que se passa no computador remoto, podendo posteriormente fazer um zoom à parte que lhe interessa mais e interagir com esta. Apesar da transferência inicial de todo o ambiente de trabalho implicar mais tráfego na ligação de dados, as actualizações posteriores tornam-se menores pois apenas são enviadas as secções do *desktop* que sofreram alterações desde a última actualização. Este efeito pode ser melhor compreendido através da **Figura IV.1**, onde no lado esquerdo se pode constatar o início de uma sessão na aplicação original e a necessidade de se navegar pelo ambiente de trabalho para obter um panorama do que este nos reserva. Inicialmente, apenas a porção rectangular do canto superior esquerdo do *desktop* se encontra disponível. Na figura do lado direito, verifica-se que na nova versão da aplicação, todo o ambiente de trabalho está automaticamente disponível ao utilizador no início de uma sessão, e o navegar pelo *desktop* apenas selecciona a porção do ambiente de trabalho à qual se pretende fazer *zoom*.

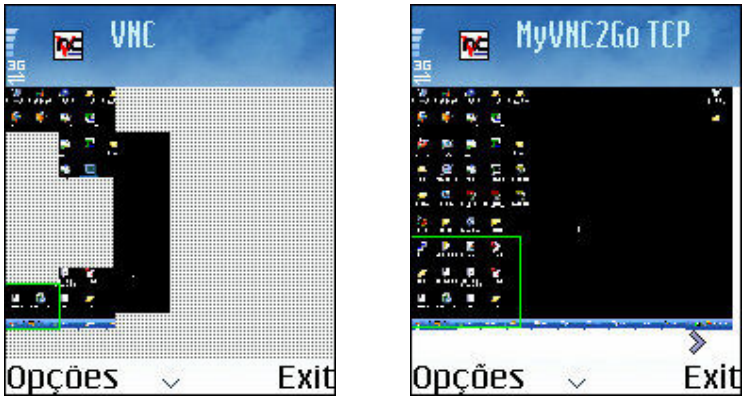
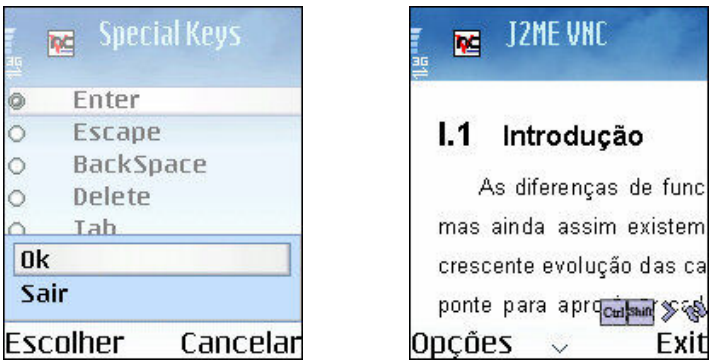


Figura IV.1 – Início de uma sessão VNC na aplicação original (à esquerda) e numa versão modificada (à direita).

2. Para permitir ao utilizador um acesso a teclas que podem ser acedidas facilmente num teclado normal de computador mas não num telemóvel, a versão original continha um modo de definir combinações de teclas (macros) que o utilizador podia utilizar. Estas macros, definidas no MIDlet *About*, apareciam directamente nos itens do menu principal durante o decorrer de uma sessão VNC, o que fazia com que a lista de itens crescesse com o aumento do número de macros disponíveis, tornando este menu pouco prático de utilizar. Para um utilizador inexperiente e não familiarizado com o formato utilizado para a definição das macros (referido no **Anexo B**), era impossível utilizar teclas que não existissem no teclado limitado do telemóvel.

   Foi criado um novo formulário, *Special keys* (**Figura IV.2 - a**), acessível pelo menu principal, com uma lista de teclas predefinidas no código da aplicação, às quais se juntam as macros definidas pelo utilizador. Basta seleccionar-se a tecla desejada neste formulário e pressionar *ok*, que esta é automaticamente enviada para o servidor. No caso de teclas como *shift, ctrl, alt left, alt right, meta,* estas são mantidas pressionadas no servidor e aparece uma indicação no ecrã do telemóvel para o utilizador não se esquecer das mesmas (este efeito pode ser visualizado na **Figura IV.2 - b**). Deste modo, podem ser efectuadas combinações de teclas para além das definidas nas macros, por exemplo, pressionar o *ctrl*, depois o *shift* (ambas ficam pressionadas no servidor até nova ordem) e depois o *escape*. Não foi realizada uma lista exaustiva de todas as teclas, mas tentou-se disponibilizar as mais comuns, sendo sempre possível ao utilizador definir mais macros.



a) **b)**

Figura IV.2 – Teclas especiais.
a) Parte do menu *Special Keys*. b) Exemplo de teclas mantidas pressionadas.

A definição de macros é efectuada no formulário *Edit Commands* disponível no MIDlet *About* (**Figura IV.3**), sendo que a sua interpretação segue o formato introduzido pelo criador da aplicação original, podendo este ser consultado no **Anexo B**.

Figura IV.3 – Formulário para definição de macros.

3. Apenas o botão esquerdo do rato e duplo clique eram disponibilizados ao utilizador. Se este pretendesse por exemplo, utilizar o botão direito do rato para aceder às propriedades de um ficheiro ou de um documento, isto não lhe era permitido. Mesmo o duplo clique nem sempre funcionava, pois, devido à diferença entre atrasos (*jitter*) provocado pela rede GPRS e a atrasos existentes na colocação das mensagens RFB na rede, chegava ao servidor não um duplo clique mas dois cliques independentes. Também não era permitido manter pressionado o botão esquerdo do rato para se poder, por exemplo, arrastar janelas ou seleccionar texto de um documento.

   A solução para resolver o problema do duplo clique passou inicialmente por enviar não dois cliques seguidos mas três. Com este truque, garantia-se que em redes GPRS o efeito obtido era o do duplo clique. No entanto, em redes UMTS ou HSDPA era efectuado um triplo clique, o que, por exemplo, num documento de texto serve para seleccionar um parágrafo completo ao invés de uma só palavra. Posteriormente, a solução encontrada (e implementada) passou por substituir as sucessivas invocações ao método que remete para o servidor o evento do rato, existente na *thread* que implementa o protocolo de comunicação RFB, para apenas uma invocação de um novo método que garante que são enviadas as mensagens necessárias ao servidor, continuamente e sem interrupções da *thread* pelo sistema operativo. É de referir que um duplo clique envolve 4 eventos de rato e logo implicava 4 invocações sucessivas ao método correspondente (pressionar do botão, libertá-lo, pressionar e libertá-lo novamente). Com o novo método implementado, a *thread* de comunicação certifica-se de que as mensagens necessárias são colocadas na rede de forma ininterrupta.

   Uma alternativa à resolução deste problema passa por introduzir uma nova mensagem ao protocolo RFB, em que se indica claramente ao servidor que se pretende efectuar um duplo clique. No entanto, esta opção traria problemas de compatibilidade com os servidores existentes, pois a nova mensagem ao não ser reconhecida por estes e sendo interpretada como um comando inválido, provocaria a terminação da sessão.

   Foram ainda simulados o botão do lado direito do rato com a tecla ‘0’ e o botão de *scrolling* com as teclas ‘3’ e ‘9’ (*scroll up* e *scroll down* respectivamente).

O duplo clique ficou na tecla '#' e o clique simples ficou na tecla '5' ou pressionando o botão de navegação do telemóvel.

O pressionar/largar a tecla esquerda do rato (diferente de clique) ficou a cargo da tecla '7' e foi associada à tecla '1' a opção de menu *Call mouse*, que coloca o rato na posição correspondente ao centro do ecrã do telemóvel. Este último ponto tem como objectivo tornar mais fácil e intuitiva uma interacção entre o modo de navegação e o modo de utilização do rato sem ser necessário aceder ao menu cada vez que se pretende colocar o ponteiro do rato na posição desejada.

No entanto, após verificação de que esta associação de teclas não funciona correctamente em telemóveis com teclados qwerty, foi criado um menu *Mouse* onde são disponibilizadas todas as opções descritas anteriormente, bastando ao utilizador seleccionar uma e pressionar *ok* (**Secção A.5.3.5**).

4. No modo em que todo o ambiente de trabalho remoto é visível no ecrã do telemóvel (*zoom out*), os pedidos de actualização só eram efectuados para a porção do *desktop* seleccionada. Isto obrigava o utilizador, no caso de ter aberto uma janela ou aplicação que ocupasse todo o ambiente de trabalho, a percorrê-lo "secção a secção" de forma a realizar a actualização na íntegra. Para melhor se visualizar este efeito, representa-se na **Figura IV.4**, um cenário exemplificativo onde se pode observar que o utilizador premiu o botão Iniciar do Windows e começou a percorrer o *desktop* de modo a tornar visível o menu expandido.

   Neste modo (*zoom out*), as actualizações passaram a ser efectuadas ao ambiente de trabalho completo e não apenas à porção seleccionada, passando a ser incrementais. Ou seja, apenas as alterações que ocorreram no *desktop* desde a última actualização, são enviados para o telemóvel. Desta forma, se o utilizador mexer o rato de posição ou abrir uma janela, apenas o ícone do rato na nova posição ou a nova janela são enviadas pelo servidor e não todo o ambiente de trabalho. Este resultado é visível no teste efectuado na **Secção V.5.3**.

Figura IV.4 – Cenário exemplificativo de uma característica da aplicação original.

5. No MIDlet *About* na versão disponibilizada, apenas era permitido adicionar novas macros e actualizar as existentes, não sendo no entanto possível apagá-las.

   Tendo isto em consideração, foi implementada a funcionalidade que permite apagar as macros, bastando para isso seleccionar a macro desejada e escolher a opção *delete*. Foi também introduzida uma limitação que não permite que sejam criadas duas macros com o mesmo nome (não faz sentido estarem disponíveis para o utilizador duas macros sem que se possa distinguir o seu conteúdo).

6. A funcionalidade do VNC que permite a realização de actualizações incrementais não era utilizada, sendo que este deve ser um ponto a optimizar para reduzir a largura de banda utilizada.

   Numa primeira abordagem ao tópico e tendo em consideração as alterações introduzidas pelos pontos 1 e 4 desta secção, todos os pedidos de actualização passaram a ser efectuados com o campo incremental diferente de zero. Ou seja, apenas as porções do ambiente de trabalho que foram alteradas desde a última actualização são enviadas pelo servidor. Existe a excepção da actualização inicial (no início da sessão) ou quando se usa explicitamente a opção de *refresh* para forçar uma actualização completa ao ecrã. Utilizando a opção de *active refresh* no telemóvel, que faz com que sejam efectuados pedidos de actualização ao servidor de 2 em 2 segundos, o facto destes pedidos de actualização passarem a ser incrementais provocou um aumento de velocidade da aplicação. Este facto era previsível dado que a quantidade de informação enviada pelo servidor numa actualização incremental é bastante inferior, comparativamente com uma actualização completa. No entanto, pode-se sempre considerar situações extremas, como por exemplo, a visualização de um filme em ecrã completo, em que as actualizações incrementais podem ter dimensões consideráveis.

7. A gestão dos sistemas anfitriões (*hosts*) era pouco prática e não permitia alterar as palavras-chave sem ter de se eliminar o sistema anfitrião em questão e criá-lo de novo.

   Alterou-se o formulário que permitia apenas visualizar e apagar os sistemas anfitriões existentes, para poder agora também, alterar a palavra-chave que lhes corresponde, sem ter de os apagar e criar de novo. Este formulário passa ainda a permitir seleccionar o sistema anfitrião de entre os existentes e efectuar a ligação ao mesmo. Antes, esta lista era visível quando se carregava no botão de menu do formulário inicial, misturada com os vários comandos (de *log*, *connect*, *manage hosts*, *exit*…), o que a tornava incómoda quando existia mais do que um sistema anfitrião armazenado em memória.

8. Se a aplicação ficasse bloqueada por algum erro interno que não fosse tratado, nada indicava ao utilizador que esta estava parada.

   Para o utilizador não ficar sem saber se o programa se encontra a funcionar ou bloqueado, foi introduzida uma simples animação de rotação no símbolo de *refresh*.

Simultaneamente, a actualização da imagem visível no ecrã do telemóvel passa a ser realizada periodicamente ao longo da actualização, ao invés de ocorrer só no fim desta. Seria interessante indicar directamente ao utilizador que ocorreu um erro e descrevê-lo, tal como acontece na maioria dos casos, mas trata-se aqui de uma alternativa para lidar com erros de causa desconhecida. No entanto, este tipo de erros não voltaram a ocorrer durante os testes efectuados à versão final.

9. Durante uma sessão de VNC estabelecida, a opção *exit* fazia com que a aplicação terminasse completamente. A ideia era substituir esta opção por um *disconnect,* para não se ter de sair da aplicação quando apenas se pode querer fechar a sessão em curso e estabelecer uma outra para outro servidor.

   Actualmente, ao se escolher a opção *exit* quando existe uma ligação estabelecida, o programa em vez de terminar por completo, volta ao formulário inicial onde se pode efectuar uma nova ligação a outro sistema anfitrião. Para terminar completamente a aplicação deve-se seleccionar novamente a opção *exit* (no formulário inicial).

10. A aplicação reconhecia quando, no servidor, era copiado algum texto para o *clipboard*, mas essa informação era simplesmente descartada.

    Passou-se a armazenar no cliente o texto copiado, permitindo ao utilizador aceder a este e modificá-lo no formulário especifico para edição de texto e envio para o servidor. Qualquer texto escrito neste formulário pode agora também ser copiado para o *clipboard* e esta informação ser passada ao servidor. Para obter este efeito foi necessário implementar as mensagens do protocolo RFB correspondentes (ver **Anexo A, secção A.5.3.6** e **A.5.4.4**).

11. Nas várias aplicações VNC testadas (descritas no **Capítulo** III.), quando em modo de utilização do cursor, se o utilizador ultrapassasse as fronteiras do ecrã, a imagem acessível do ambiente de trabalho deslocava-se automaticamente para ser visível a nova posição do rato. Isto permitia utilizar o cursor e navegar pelo ambiente de trabalho sem ter de se alternar entre o modo de utilização do cursor e o modo de navegação. No entanto esta característica não se encontrava disponível na aplicação escolhida.

    Foram efectuadas as alterações necessárias para se conseguir obter este efeito no cliente do telemóvel, tendo em consideração os diferentes modos de navegação e a utilização dos diferentes factores de escala (*zoom*) existentes.

12. Para telemóveis com teclados qwerty, as teclas que permitem alternar entre os modos de navegação, cursor, SMS e escrita, assim como as que permitem modificar o factor de escala, não funcionam.

    Para solucionar isto, foi adicionado um novo menu, *Change Mode,* onde se pode escolher o modo desejado e fazer *zoom in* e *zoom out*, sem o recurso às teclas predefinidas.

Como a maioria destes telemóveis têm também um estilete (*stylus*) que pode ser utilizado para mexer o cursor do rato, independentemente do modo seleccionado, passou também a ser permitido no modo de escrita, navegar pelo ambiente de trabalho com as teclas de navegação do telemóvel. Deste modo, para utilizadores de telemóveis com estilete e teclados qwerty, é possível navegar pelo ambiente de trabalho, mexer o cursor do rato e escrever utilizando o teclado integrado, tudo sem ter de alternar entre modos, como acontece nos telemóveis mais usuais.

13. No caso de perda ou furto do telemóvel, os endereços dos sistemas anfitriões e correspondentes palavras-chave armazenadas na aplicação, davam a quem tivesse acesso ao telemóvel, a possibilidade directa de os controlar remotamente.

    Para evitar tal situação, tornando mais segura a aplicação, foi introduzido um mecanismo de segurança que passa pela utilização de uma palavra-chave a restringir o acesso ao formulário do livro de endereços. Deste modo, dentro deste formulário existe a opção *Set Master Pass* que permite definir a palavra-chave para acesso ao mesmo. No caso de nenhuma palavra-chave ser introduzida, este mecanismo é desactivado e qualquer utilizador terá acesso directo ao livro de endereços.

    Este mecanismo apresenta no entanto um nível de segurança bastante simples de contornar, pois apesar de proteger o acesso alheio aos sistemas anfitriões armazenados, de indivíduos sem conhecimentos da aplicação, é possível, através do acesso ao sistema de ficheiros do telemóvel e sabendo o que procurar, aceder a toda esta informação que se encontra armazenada num ficheiro criado pela API RMS (*Record Management System*) do Java.

IV.2 Modos de navegação e alterações ao factor de escala

Sendo este um ponto importante a ter em consideração dado que os ecrãs dos telemóveis são de dimensões reduzidas, foram encontrados melhoramentos ao método existente, de forma a tornar mais intuitiva, fácil e apelativa a navegação e as alterações ao factor de escala (*scaling*) utilizados na aplicação. Fala-se em navegação, porque o ecrã de um telemóvel apenas acomoda uma pequena porção do ambiente de trabalho de um computador comum e por isso é necessário permitir que se navegue por todo este, para o utilizador ter acesso total sobre o mesmo.

No cliente original, apenas existiam dois factores de escala, ou zoom's, utilizados. Um deles acomodava todo o ambiente de trabalho do servidor, reduzido de um factor de escala suficiente para que fosse visível por inteiro no ecrã do telemóvel (*zoom out*). O outro utilizava um factor de escala unitário, o que fazia com que a porção da imagem visível no telemóvel correspondesse a uma porção do ambiente de trabalho com exactamente as dimensões do ecrã do telemóvel (*zoom normal*).

A navegação no modo de *zoom out* era realizada pelo deslocamento de uma pequena secção, com o objectivo de seleccionar a área à qual se deseja alterar o factor de escala. Todas as passagens para o modo de *zoom normal* implicavam um pedido de actualização ao servidor pois o

cliente apenas guardava em memória uma cópia do ambiente de trabalho em *zoom out*. Para percorrer todo o ambiente de trabalho do servidor em modo *zoom normal*, visto que apenas era guardado em memória uma porção deste correspondente às dimensões do ecrã do telemóvel, cada movimento, em qualquer direcção, implicava um pedido de actualização ao servidor de uma parcela do ambiente de trabalho correspondente ao deslocamento efectuado. Assim, se um movimento para a direita implica um deslocamento de X *pixeis* nesta direcção, a imagem visível no telemóvel sofrerá um deslocamento igual para a esquerda, revelando um rectângulo branco com X *pixeis* de largura e a ocupar toda a altura do ecrã do telemóvel. Seguidamente é efectuado um pedido de actualização ao servidor, correspondente à porção do ambiente de trabalho que falta para completar a imagem visível na nova posição. Este efeito pode ser observado na **Figura IV.5**.

Figura IV.5 – Navegação pelo ambiente de trabalho em *zoom normal* na aplicação original.

IV.2.1 Alterações efectuadas

Apesar deste método requerer pouca memória no telemóvel, para navegações intensivas pelo ambiente de trabalho o acesso à rede é considerável e constante e não é apelativo ao utilizador. Deste modo, foi criado um novo método para efectuar alterações ao factor de escala da imagem visível e foi criado um novo modo de navegação. Para além disso foi ainda introduzido um nível de *zoom* em que o ambiente de trabalho é reduzido para metade, 50% (*zoom fifty*).

Numa primeira fase, passaram a existir *buffer's* em memória para armazenar todo o ambiente de trabalho nos 3 zoom's diferentes (visíveis na **Figura IV.6**). Um deles, consegue acomodar as dimensões reais do ambiente de trabalho do servidor (*zoom normal*), um outro acomoda metade destas dimensões (*zoom fifty*) e um terceiro acomoda somente as dimensões do ecrã do telemóvel pois todo o ambiente de trabalho é reduzido até ser visível por completo neste (*zoom out*). A ideia é, após efectuada uma primeira actualização total ao ambiente de trabalho, poder-se navegar por todo este (incluindo mudar de factor de escala), sem serem realizados novos pedidos de actualização.

Figura IV.6 – Representação dos 3 *zoom's* disponíveis na aplicação.
Da esquerda para a direita: *zoom out, zoom fifty* e *zoom normal*.

Na aplicação original existiam apenas dois *buffer's* com as dimensões do ecrã do telemóvel (um por cada *zoom*), mas enquanto o correspondente ao *zoom out* permitia armazenar todo o ambiente de trabalho reduzido ao ponto de ser visível no ecrã do telemóvel, o correspondente ao *zoom normal* apenas armazenava uma porção do ambiente de trabalho com as dimensões do ecrã do telemóvel. Isto fazia com que a navegação pelo ambiente de trabalho implicasse pedidos de actualizações constantes. Com esta nova implementação, realizou-se uma troca de quantidade de dados transferidos e tempo gasto ao navegar pelo ambiente de trabalho, por memória necessária no telemóvel e uma navegação mais suave e fluida.

A navegação é efectuada utilizando o botão de navegação ou as teclas 2, 4, 6 e 8 e em qualquer altura se pode fazer mais *zoom* com a tecla 3 ou menos *zoom* com a tecla 9 para alternar entre os três factores de escala disponíveis.

Foi ainda introduzida a possibilidade de o utilizador poder optar por ter as alterações ao factor de escala realizadas do lado do servidor, bastando para isso seleccionar essa opção no formulário inicial e garantir que o servidor alvo suporta esta funcionalidade. Caso isto não se verifique, a ligação será terminada e dever-se-á utilizar o modo em que as alterações ao factor de escala são processadas no telemóvel. Ao estabelecer a sessão de VNC, o cliente envia um pedido para definir o factor de escala a utilizar, caso o servidor não reconheça esta mensagem (não faz parte do protocolo RFB original), enviará uma mensagem de erro que o cliente utiliza para terminar a sessão e informar o utilizador.

Esta opção permite vários factores de escala (1:1, 1:2, 1:3… até todo o ambiente de trabalho ser visível no ecrã do telemóvel), bastando pressionar as teclas 3 e 9 para fazer *zoom in* e *zoom out* respectivamente. Ao contrário do modo em que as alterações ao factor de escala são processadas no telemóvel, neste caso, por cada *zoom* efectuado, é realizado um pedido de actualização ao servidor. Este, por sua vez, envia como resposta o ambiente de trabalho já afectado pelo novo factor de escala. No entanto, a navegação sem alterar o *zoom* mantém-se inalterada, ou seja, pode-se percorrer todo o ambiente de trabalho sem serem efectuados novos pedidos de actualização.

Mesmo considerando que esta funcionalidade implica a utilização de um *buffer* no servidor, que será tanto maior quanto maior for o ambiente de trabalho a armazenar, e que este coloca algumas questões negativas de performance e de utilização de memória, como já existe pelo menos um servidor que disponibiliza esta funcionalidade (*WinVNC Version 3.3.3r7 with server side scaling*

extensions), resolveu-se oferecer suporte à mesma. Apesar de hoje em dia os computadores terem capacidades de memória cada vez maiores e processadores cada vez mais rápidos, o VNC assenta no pressuposto de ser um sistema portável e compatível com o maior número de máquinas possíveis e por isso a exploração deste caminho não é recomendada para servidores VNC de uso genérico. No entanto, é preciso ter em consideração o âmbito deste trabalho e que a alternativa é colocar estes aspectos negativos no lado do cliente, que no nosso caso é bastante mais limitado (telemóvel vs PC).

A extensão ao protocolo utilizada resume-se a uma nova mensagem da parte do cliente para o servidor a pedir uma alteração ao factor de escala, cujo formato se apresenta na **Tabela IV.1**. As mensagens utilizadas pelo protocolo RFB usam os tipos básicos U8, U16, U32, S8, S16, S32, que representam respectivamente 8, 16 e 32-bit *unsigned integer* e 8, 16 e 32-bit *signed integer*.

Tabela IV.1 – Formato da mensagem de pedido de mudança de factor de escala.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	F	*tipo da mensagem*
1	U8		*Factor de escala*
2			*enchimento com zeros (padding)*

O servidor ao receber um pedido de alteração ao factor de escala, responde com uma mensagem indicando as dimensões do ambiente de trabalho e as dimensões do novo *buffer* depois de efectuada a alteração, como pode ser visto na **Tabela IV.2**. Posteriormente, o servidor transfere para o cliente o *desktop* já afectado pelo novo factor de escala.

Tabela IV.2 – Formato da mensagem indicando uma alteração ao factor de escala.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	F	*tipo da mensagem*
1			*enchimento com zeros (padding)*
2	U16		*largura do desktop*
2	U16		*altura do desktop*
2	U16		*largura do buffer*
2	U16		*altura do buffer*
2			*enchimento com zeros (padding)*

Durante o desenvolvimento da aplicação e à medida que se foram realizando testes à mesma, verificou-se que os telemóveis disponíveis não suportavam aceder a ambientes de trabalho com dimensões consideráveis, pois a aplicação encerrava-se autonomamente, por falta de memória. Visto que a quantidade de memória disponível na maioria dos telemóveis é bastante limitada e como a navegação mais fluida pelo ambiente de trabalho implica o armazenamento de todo o ambiente de trabalho na memória do dispositivo, dependendo do modelo de telemóvel utilizado e da resolução do ambiente de trabalho, era simples conceber uma combinação destes factores, em que o telemóvel acusasse falta de memória. Dado que actualmente, cada vez mais os computadores pessoais apresentam resoluções maiores (para acompanhar o aumento do tamanho dos monitores), tornou-se indispensável deixar de parte a ideia inicial de ter um só método de navegação, melhorado mas que requer mais memória, revelando-se necessário encontrar uma alternativa mais abrangente ao mesmo

ou permitir ao utilizador escolher o método a utilizar de acordo com as características das máquinas em questão (cliente e servidor).

Deste modo, considerando que os telemóveis vão tendo cada vez mais capacidade de memória e que a fluidez com que se navega pelo ambiente de trabalho sem a necessidade de realizar actualizações constantes, é um factor importante e apelativo ao utilizador, resolveu-se manter disponível a alternativa descrita anteriormente e acrescentou-se uma nova, dando ao utilizador a possibilidade de optar entre ambas. A alternativa encontrada passou por, reduzir o tamanho do buffer de memória utilizado no telemóvel para armazenar o *zoom normal* (100%), que em vez de ter capacidade para armazenar todo o ambiente de trabalho, passou apenas a armazenar uma porção deste das dimensões do ecrã do telemóvel. Voltou-se assim ao modo original de como era efectuada a navegação, ou seja, voltou a ser necessário, no modo de *zoom normal*, fazer pedidos de actualização constantes, à medida que se navega pelo ambiente de trabalho. No entanto, o modo de *zoom fifty* (50%) manteve-se inalterado, conseguindo-se manter uma navegação fluida e sem necessidade de pedidos de actualização. Tal revelou-se possível porque a quantidade de memória ocupada por este *buffer* é inferior à ocupada pelo *zoom normal* (ver **Secção V.5.5**), não se tendo verificado que os telemóveis testados acusassem falta de memória.

O modo mais afectado com esta modificação é quando a alteração ao factor de escala é efectuado pelo servidor. Neste caso, como apenas é utilizado um *buffer* com as dimensões do ecrã do telemóvel para armazenar as actualizações recebidas, e como as alterações ao factor de escala, implicam comunicação com o servidor, é necessário efectuar pedidos de actualização constantes à medida que se navega pelo ambiente de trabalho.

Com estas modificações ficaram disponíveis 4 modos distintos de navegação e alteração ao factor de escala, tentando-se abranger os vários tipos de utilizadores e os vários modelos de telemóveis existentes:

- **CS (client side) Scaling com Smooth Nav:** Modo que requer mais memória disponível da parte do telemóvel mas que será provavelmente o mais apelativo ao utilizador pela sua navegação suave e fluida.
- **CS Scaling sem Smooth Nav:** Requer menos memória que o modo anterior, mas perde o sentido de navegação suave e fluida quando em *zoom normal*, mantendo-o apenas em *zoom fifty*. A navegação em *zoom normal* implica pedidos de actualização constantes.
- **SS (server side) Scaling com Smooth Nav:** Segundo modo que requer mais memória permitindo também uma navegação suave. No entanto são efectuados pedidos de actualização quando se procede a alterações ao factor de escala. Este modo requer menos capacidade de processamento da parte do telemóvel.
- **SS Scaling sem Smooth Nav:** É o modo que exige menos do telemóvel, requerendo pouca memória e pouco processamento. Útil para telemóveis mais antigos e limitados mas implica actualizações constantes durante a navegação e alterações ao factor de escala, o que se pode tornar incomodo para o utilizador, devido à latência da rede.

IV.3 Transferência de Som

Para permitir um controlo pleno sobre o computador alvo e para se ter a percepção de se estar efectivamente sentado à frente do PC, não é apenas necessário visualizar o seu ambiente gráfico mas é também essencial ouvir tudo o que se passa neste. Dependendo de como o utilizador configurar o computador a ser controlado, este pode permitir que os clientes tenham acesso ao áudio que vai para a placa de som (sons do Windows, música a tocar, etc.), ou em alternativa, transferir todo o áudio exterior ao computador que seja capturado pelo microfone. Em alguns casos pode-se obter uma mistura de ambos os dispositivos de entrada, sendo para isso necessário uma placa de som que o possibilite.

Nas versões actuais do VNC (*RealVNC, TightVNC e UltraVNC*), o único som que é reproduzido no cliente é um *beep* referente, por exemplo, a um erro que ocorra na linha de comandos. Ou seja, um evento no servidor despoleta o envio de uma mensagem para o cliente, que por sua vez, faz com que este provoque um som para informar o utilizador de que algo errado se passa.

Sendo que nenhuma das soluções existentes em VNC, nem o próprio protocolo RFB, permite a transferência de som entre o servidor e cliente (apesar de ser um tópico interessante e pedido pelos utilizadores nos fóruns dedicados ao assunto), duas soluções para solucionar esta lacuna foram identificadas:

- Introduzir uma extensão ao protocolo para acomodar a transferência de sons associados a eventos. Desta forma, caso um evento ocorra, o servidor envia uma mensagem ao cliente contendo uma cadeia de caracteres com o nome do evento. Se o cliente tiver conhecimento de que som tocar para esse evento particular, toca-o, se não souber, envia uma resposta ao servidor pedindo-lhe que envie o ficheiro com o som a reproduzir (formava-se assim uma espécie de *cache* no cliente).

  Esta solução encontra no entanto alguns reveses. Para além do protocolo RFB existente apenas permitir extensões ao nível de codificações, pseudo-codificações e tipos de segurança (ver **Anexo A**, secções **A.4**, **A.5.2**, **A.5.5**, **A.5.6**), fazendo com que a criação de novas mensagens possa por em risco a compatibilidade com clientes e servidores já existentes, também o facto desta alternativa apenas permitir transferir sons relacionados a eventos e não todo o áudio disponível no servidor (música, sons exteriores capturados pelo microfone, etc), fez com que esta solução fosse descartada.
- Utilizar uma ligação paralela à sessão RFB, especificamente criada para transferir o som.

  A ideia é fazer *streaming* de todo o áudio que chega à placa de som através de uma sessão dedicada, do servidor para o cliente. Com esta alternativa levantaram-se algumas questões pertinentes, nomeadamente como se comportará a mesma com vários clientes em simultâneo e se esta transferência de som deve ser unidireccional ou bidireccional, para permitir por exemplo, caso se esteja a efectuar uma assistência remota, que o auxiliado e o auxiliando possam comunicar entre si. Descreve-se em seguida esta solução.

A solução adoptada passa por transferir todo o áudio do servidor para o telemóvel, através de uma sessão paralela à utilizada pelo VNC, dedicada para o efeito. Podem ser os sons do Windows que vão para a placa de som, música, o som capturado pelo microfone ou uma mistura de tudo,

dependendo das capacidades da placa de som do computador alvo e do que estiver seleccionado nas opções de gravação de áudio do sistema operativo. Esta selecção tem de ser realizada pelo utilizador nas configurações de áudio do sistema, pois é dependente da placa de som e por consequente, não foi possível fazer com que o servidor de VNC activasse automaticamente a captura de áudio pretendida.

Dado que este trabalho é especifico para clientes em telemóveis e que apenas dispomos de uma API (e hardware) limitada, para manter a compatibilidade com um maior número de plataformas possíveis, resolveu-se não utilizar nenhum protocolo de *streaming* RTP/RTSP (RFC 3550/2326), pois estes apenas são suportados nos *player's* nativos da maioria dos telemóveis e apenas modelos mais recentes e topo de gama suportam-nos sobre as API's do J2ME. Isto significa que a maioria dos telemóveis apenas permite reproduzir áudio (ou mesmo vídeo) em blocos, depois de os carregar na íntegra para memória, e portanto, outra solução teve de ser equacionada para reproduzir o efeito de *streaming* desejado. Deste modo, resolveu-se fazer uma análise aos protocolos de comunicação possíveis de utilizar, nomeadamente o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*), com o intuito de escolher o que melhor se adapta à situação em questão.

IV.3.1 TCP *vs* UDP

Apesar de ambos os protocolos serem bastante conhecidos, algumas características específicas são introduzidas pelas redes GPRS/UMTS e por isso segue-se uma breve descrição e análise de cada um deles.

IV.3.1.1 TCP (Transmission Control Protocol)

Definido no RFC-793, o TCP é o protocolo mais comum na Internet e mais utilizado para transferência de dados sem falhas. Isto deve-se principalmente ao facto do TCP oferecer protecção contra erros, garantia de entrega dos dados pela ordem por que são enviados e oferece ainda um mecanismo de controlo de congestão. O TCP é um protocolo orientado à conexão responsável pela comunicação fiável entre dois equipamentos, interligados através de uma rede IP. Os dados são transferidos numa sequência de bytes (*stream*), sendo que cada conjunto de bytes de dados, precedidos de um cabeçalho TCP, é chamado de segmento.

Sendo um protocolo orientado à conexão, significa que antes de transmitir dados, é necessário estabelecer a sessão entre os dois pontos de comunicação. A transferência de informação pode ser efectuada em ambos os sentidos, e quando esta termina, é necessário terminar a sessão para libertar os recursos do sistema. Ambos os terminais sabem quando uma sessão é iniciada e terminada e ambos podem iniciar ou terminar uma ligação.

Sendo a transferência de dados efectuada numa sequência de bytes (*stream*), não existem fronteiras na informação transferida, ou seja, os segmentos enviados podem ser fragmentados, ou agrupados a outros, dependendo do nível de carga da rede, dos algoritmos de comutação utilizados, de erros aleatórios e de outros parâmetros que não se podem controlar. A única garantia oferecida é

que todos os dados serão recebidos no destino sem erros e na ordem correcta. Na eventualidade de surgirem erros durante a transmissão de dados, os segmentos afectados serão retransmitidos.

Na **Figura IV.7** pode-se ver o formato do cabeçalho de uma trama TCP.

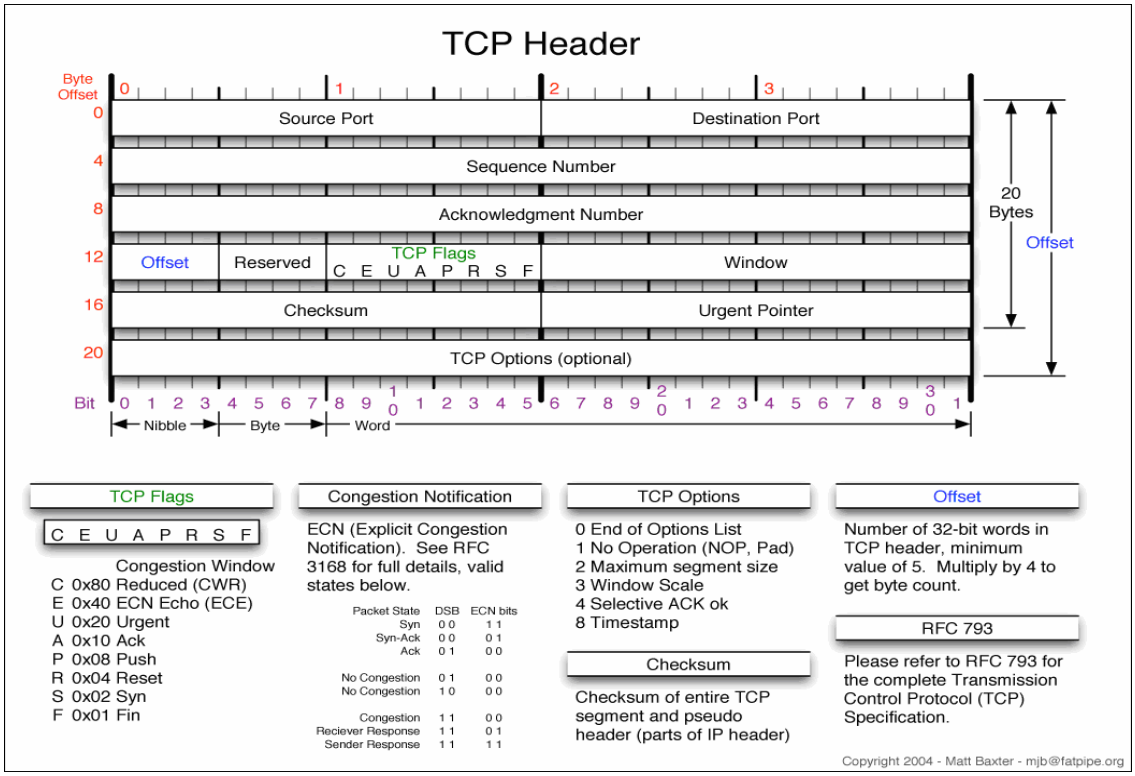


Figura IV.7 – Cabeçalho de uma trama TCP

IV.3.1.1.1 Vantagens do TCP

- Como está incorporado no Sistema Operativo, gerir os segmentos recebidos, implica menos mudanças de contexto do núcleo do sistema para o espaço do utilizador, pois todo o processo de confirmação de segmentos, controlo de congestão, junção de segmentos, etc, é efectuado no núcleo do sistema.
- O TCP garante que os dados chegam ao destino, que são entregues por ordem e que não existe duplicação dos mesmos.

IV.3.1.1.2 Desvantagens do TCP

- O protocolo pode estar optimizado para condições diferentes daquelas que se vão encontrar. Neste caso, os protocolos utilizados pelo GRPS já por si garantirem a entrega de pacotes, ou seja, esta característica do TCP torna-se redundante. A especificação do GPRS define protocolos para o plano de transmissão e para o plano de sinalização. Entre eles encontram-se o LLC (*Logical Link Control*) e o RLC (*Rádio Link Control*), sendo que o primeiro provê uma ligação lógica bastante fiável entre a MS (*Mobile Station*) e a SGSN (*Serving GPRS support nodes*) associado a ela. Suas funcionalidades incluem controle de sequência, entrega

em ordem, detecção e correcção de erros e retransmissão. O RLC tem como principal objectivo estabelecer uma ligação fiável entre a MS e a BSS (*Base Station System*). Entre as suas funções encontram-se, fragmentação e reconstrução dos quadros LLC em blocos de dados RLC, e correcção de erros através de um mecanismo de retransmissão selectiva desses blocos. Apesar disto, pode existir perda de pacotes numa ligação GPRS mas a sua incidência é relativamente rara e difícil de quantificar [8] [9] [10].

- Não tem fronteiras nos pacotes recebidos. Ou seja, como referido anteriormente, pode receber vários segmentos juntos, quando estes foram enviados separados, ou pode receber um único segmento quando foram enviados vários. É preciso criar essas fronteiras para se poder fazer uso da informação recebida.
- Não pode ser utilizado para *broadcast* ou *multicast*.

IV.3.1.2 UDP (User Datagram Protocol)

Definido no RFC-768, o protocolo UDP oferece, ao contrário do TCP, uma ligação entre dois terminais não orientada à conexão, não oferecendo correcção de erros, garantia de entrega, nem garantia de entrega por ordem dos dados transmitidos. Deste modo, consegue ser mais rápido que o TCP, pois só envia os dados, não se preocupando com mais nada. O UDP tem um *overhead* mínimo, sendo cada pacote composto por um cabeçalho e por dados do utilizador. A este pacote dá-se o nome de datagrama.

Por não ser orientado à conexão, podem-se enviar datagramas em qualquer instante no tempo, sem ser necessário avisar o destino, negociar uma ligação ou preparar seja o que for previamente. Ao contrário do TCP, os datagramas UDP tem as suas fronteiras bem definidas, ou seja, se um terminal enviar um datagrama com um determinado conteúdo, será recebido no destino, apenas um datagrama contendo o mesmo conteúdo (ou eventualmente o datagrama perde-se e não se recebe nada), não havendo ao nível da camada de transporte, segmentação de mensagens na origem ou fusão de datagramas no destino.

Apesar do UDP ser um protocolo não fiável, a taxa de falhas é bastante reduzida na Internet, sendo no entanto mais alta para redes sem fios onde mais facilmente ocorrem colisões e interferências.

Todos os mecanismos de controlo de congestão, confirmações de entregas, transacções, etc, são da responsabilidade do programador, só necessitando de serem implementadas aquelas características que realmente são necessárias.

Na **Figura IV.8** pode-se observar o formato do cabeçalho de uma trama UDP.

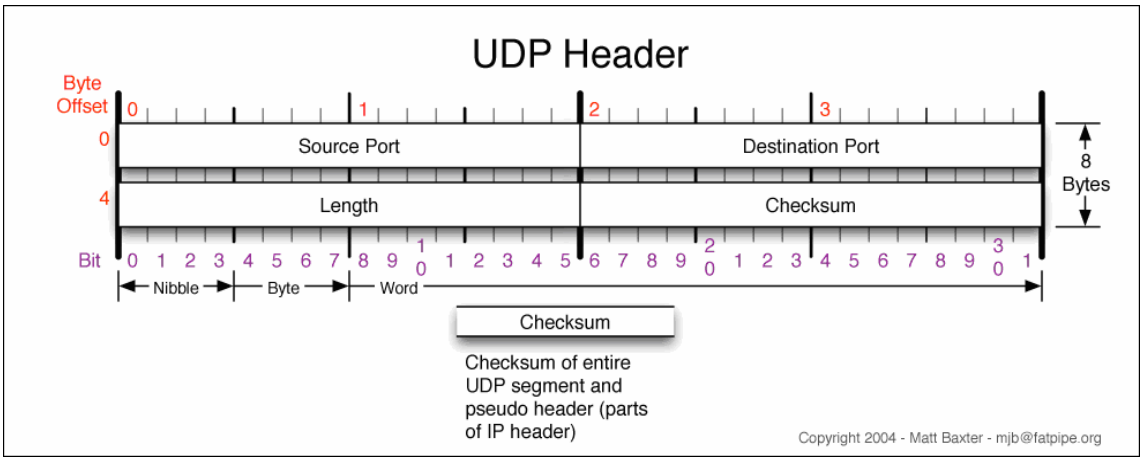


Figura IV.8 – Cabeçalho de uma trama UDP

IV.3.1.2.1 Vantagens do UDP

- O *overhead* do Sistema Operativo é menor do que o do TCP assim como a latência no arranque de uma comunicação, pois não existe estabelecimento de sessão nem o mecanismo de arranque lento (*slow-start*) associado ao TCP.
- O receptor de um pacote UDP, recebe-o exactamente como foi enviado, com as suas fronteiras bem definidas.
- O UDP permite transmissões em *broadcast* e *multicast*.
- Com o UDP pode-se enviar/receber dados num mesmo porto/*socket* de várias máquinas distintas, pois o endereço de destino é especificado por cada chamada ao sistema para enviar dados.

IV.3.1.2.2 Desvantagens do UDP

- Não existem garantias. Um pacote pode não ser entregue, ser entregue repetidas vezes, ou entregue fora de ordem. O emissor não recebe indicação nenhuma do que se passa no receptor a menos que este decida informá-lo (tenha sido programado para tal).
- O UDP não tem mecanismo de controlo de congestão. A implementação de tal mecanismo cabe ao programador caso assim o deseje. Em alternativa, pode-se optar pelo uso do DCCP (*Datagram Congestion Control Protocol*) definido no RFC-4340. O DCCP é um protocolo da camada de transporte orientado à mensagem como o UDP mas que providencia às aplicações mecanismos de controlo de congestão *standards*, sem ser necessário implementá-los na camada da aplicação.
- Os datagramas UDP são os primeiros a serem descartados quando um comutador está com falta de memória.

IV.3.1.3 Conclusões Gerais sobre TCP vs UDP

Quando uma ligação de rede é boa, o *overhead* introduzido pelo TCP é suficientemente diminuto para ser ignorado. Quando esta ligação é fraca, o *overhead* introduzido pelo TCP é significativo mas ao mesmo tempo a perda de pacotes do UDP também se torna um problema sério, o que implica retransmissões para garantir que as mensagens cheguem ao destino. Várias estudos foram efectuados sobre a performance do TCP e UDP sobre as redes GPRS e UMTS, sendo que alguns dos documentos resultantes destes estudos podem ser visualizados em [8], [9], [10] e [11]. Na comparação de resultados entre o TCP e o UDP, observa-se que o tamanho dos pacotes influencia o desempenho do TCP sobre GPRS. Para o tamanho padrão (536 bytes), o TCP apresenta débitos próximos dos alcançados pelo UDP. Para pacotes de 1500 bytes o débito atingido revela-se inferior ao débito obtido pelo UDP [8]. Estes resultados dependem do tipo de codificação utilizada pela rede e pela versão do TCP utilizada.

IV.3.2 Implementação

Tendo em consideração os prós e contras de ambas as alternativas (utilizar o protocolo TCP ou o protocolo UDP), resolveu-se implementar a transferência de som sobre ambos os protocolos de comunicação, para se poder realizar uma análise comparativa e assim seleccionar a que apresente melhores resultados.

Para este efeito, foram efectuadas alterações tanto do lado do servidor como do lado do cliente, mantendo-se sempre a compatibilidade com os clientes e servidores de VNC existentes.

A análise dos resultados obtidos por ambas as implementações pode ser vista na **Secção V.5**.

IV.3.2.1 Alterações no lado do servidor

Incorporou-se no servidor uma nova *thread*, que inicialmente abre os dispositivos de captura de áudio da API do Windows e posteriormente fica bloqueada à espera num *socket* que algum cliente se ligue. Se nenhum cliente se ligar a este *socket*, o servidor de VNC procederá como se não tivesse sido alterado, garantindo deste modo, a compatibilidade com dispositivos que não exijam som. A ideia por detrás do que foi dito no parágrafo anterior pode ser visualizada na **Figura IV.9**, onde se constata que a transferência de som é efectuada separadamente da comunicação VNC e que poderá existir uma sessão VNC estabelecida sem correspondente sessão de transferência de som. O contrário, no entanto, já não se verifica.

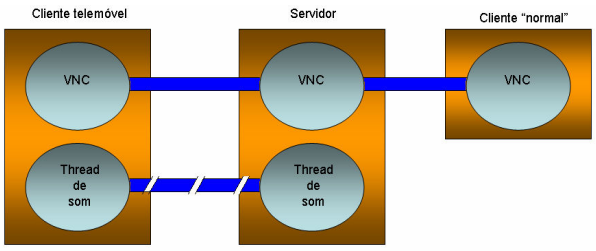


Figura IV.9 – Incorporação do mecanismo de transferência de som.

O porto associado à transferência de som é, por comodidade, o porto imediatamente a seguir ao utilizado pelo VNC e se este último for alterado durante a execução do servidor, o *socket* associado ao som é fechado e reaberto no novo porto e a *thread* correspondente é terminada e é iniciada uma nova *thread* (comportamento análogo ao efectuado pelo *socket* e *threads* de comunicação usados pelo VNC). Deste modo, garante-se que durante este processo, todos os clientes com uma ligação de áudio são correctamente terminados. O *socket* ao ser fechado liberta a *thread* que está bloqueada no *accept*, podendo esta terminar a ligação a eventuais clientes ligados e fechar-se. A nova *thread* então criada já estará associada ao novo *socket*.

Se algum cliente efectuar uma ligação para transferência de áudio, o servidor cria um cabeçalho no de 44 bytes no formato WAVE (como pode ser visualizado na **Figura IV.10** [5]) e envia-o ao cliente. Este utiliza a informação nele contida para dimensionar o *buffer* de recepção e *buffer's* auxiliares, para deste modo, utilizar apenas o mínimo de memória necessária. O cabeçalho é posteriormente cedido aos *players* que carecem desta informação. O formato do cabeçalho foi escolhido para manter uma compatibilidade com um maior número de dispositivos possíveis.

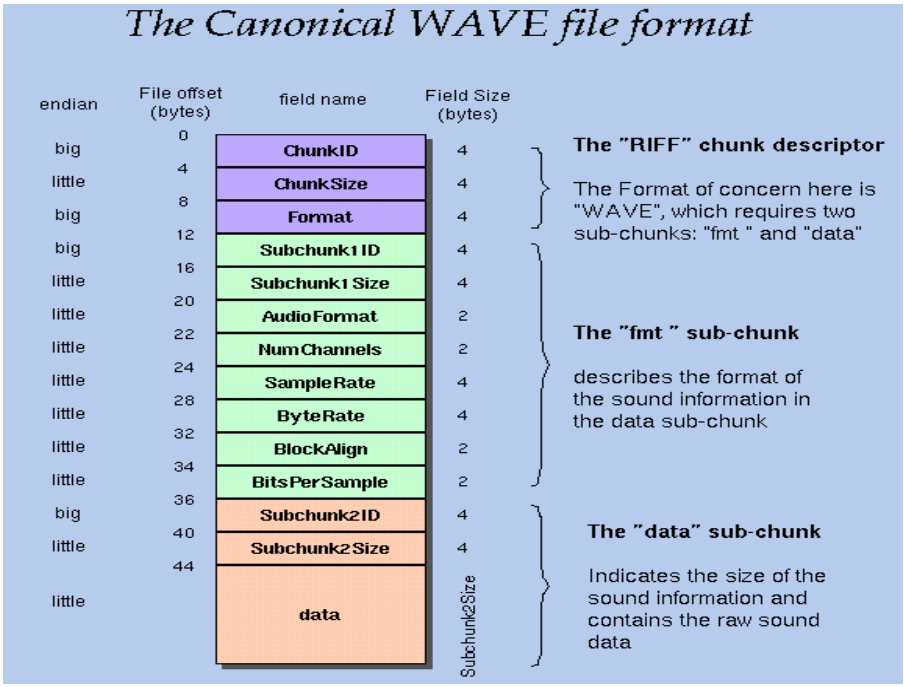


Figura IV.10 – Formato de um ficheiro WAVE [5]

Depois do cliente estar preparado para receber os pacotes de áudio, o servidor transfere todo o áudio recebido da placa de som, em PCM, 8000 bytes por segundo, mono, para consumir o mínimo de largura de banda possível. Existe ainda a possibilidade de compressão dos pacotes enviados, utilizando um algoritmo de supressão de símbolos repetidos que permite também, a realização de uma supressão de silêncio bastante simples. O contributo oferecido por estes métodos para a redução da largura de banda utilizada pode ser observado na **Secção V.5.1**, encontrando-se os mesmos descritos na **Secção IV.3.3**. Este tipo de compressão e supressão de silêncio foram escolhidos para diminuir a largura de banda ocupada pela transferência de som, tendo em consideração que a maioria dos algoritmos existentes para este efeito são proprietários, e

considerando que uma compressão complexa exigiria um descompressão também ela complexa no hardware limitado dos telemóveis.

No cliente, associa-se o cabeçalho WAVE inicialmente recebido aos pacotes de áudio, para poderem assim ser reproduzidos pela maioria dos *players* existentes nos telemóveis. No servidor, utiliza-se um sistema de *double buffering* para a captura de som, para permitir que um dos *buffer's* esteja a ser preenchido com uma nova amostra de áudio, enquanto o conteúdo do outro está a ser codificado e enviado para o cliente. Deste modo, elimina-se o risco de provocar atrasos no *driver* do som, por este não ter um *buffer* disponível para armazenar o áudio que vai capturando.

Este sistema funciona até um máximo de `MAX_CLIENTS` clientes (definido a 128 no VNC) para permitir que o ambiente de trabalho seja partilhado por vários utilizadores (funcionalidade do VNC) e que todos possam ter acesso ao áudio.

O primeiro cliente a estabelecer uma ligação para transferência de áudio faz com que o servidor reduza o volume do som do Sistema Operativo. Isto é necessário para o áudio ser perceptível no telemóvel, sendo transferido num nível bastante reduzido para posteriormente ser amplificado pelo terminal. Caso contrário, só era audível ruído nos telemóveis testados. Esta solução foi seleccionada pela sua simplicidade, por não exigir processamento adicional dos pacotes de áudio e por poder ser desejável a redução do volume de som no servidor, para não incomodar eventuais indivíduos na proximidade do mesmo. O volume de som do sistema operativo é restabelecido ao valor original depois do último cliente com uma ligação para transferência de som terminar a sua sessão. A captura de som no servidor também só é efectuada enquanto algum cliente se encontrar ligado com uma sessão para transferência de som.

Como se pode perceber pela descrição anterior, um cliente passa a ter uma ou duas ligações estabelecidas com o servidor, sendo que uma é a sessão estabelecida pelo VNC e a outra é a sessão para transferência de som, podendo esta última nunca ser iniciada pelos clientes, ou então, ser iniciada e interrompida várias vezes durante uma mesma sessão VNC.

IV.3.2.1.1 Diferenças entre a solução em TCP e a solução em UDP

Na **solução em TCP**, quando um cliente se liga ao *socket* específico que está à escuta de ligações, a *thread* cria um novo *socket* num porto aleatório, com o qual estabelece a sessão de comunicação com o cliente. Este novo *socket* é guardado num vector do tipo `SOCKET viewer[MAX_CLIENTS]`, contendo a informação de todos os *socket's* associados a clientes com sessão estabelecida.

Quando é capturado som, este, depois de comprimido, é enviado para todos os clientes ligados, utilizando-se para tal, a informação contida no vector de *socket's* associados a clientes. Juntamente com o pacote de áudio, são enviados mais 4 bytes contendo a dimensão do pacote a que estão associados. Isto torna-se necessário, pois com a compressão utilizada, os pacotes não têm todos o mesmo tamanho e sendo que em TCP os dados são enviados numa sequência de bytes (*stream*), é necessário para o cliente saber como separá-los para os poder depois descodificar.

O som é transferido em pacotes de 4 segundos (32000 bytes ou menos se comprimido) sendo este facto explicado na implementação do cliente na **Secção IV.3.2.2**.

Na **solução em UDP**, como não existe estabelecimento de sessão, é necessário que o cliente coloque na rede uma mensagem que informe o servidor de que pretende receber som. Esta mensagem é apenas um datagrama contendo o byte '1' no campo de dados. O servidor ao receber o pedido de ligação, armazena o endereço do terminal que efectuou a solicitação num vector de estruturas do tipo `struct sockaddr_in viewer[MAX_CLIENTS]`, para de uma forma análoga ao que acontece na solução em TCP, poder transferir o som capturado para todos os clientes que estão à espera de o receber.

No caso do UDP, não é necessário enviar para o cliente o tamanho do pacote de áudio, pois os datagramas UDP têm as suas fronteiras bem definidas. Deste modo, o cliente ao receber um datagrama sabe que aquele foi exactamente o pacote transmitido.

Ao receber uma mensagem contendo o byte '0', o servidor sabe que o cliente não pretende continuar a receber áudio e portanto, pode apagar o seu endereço do vector que contém os clientes com sessão estabelecida para transferência de som.

O áudio, no caso do UDP, é enviado em pacotes de 62,5 ms (500 bytes ou menos se comprimido), pois existe uma limitação à dimensão máxima dos datagramas transferidos. Esta limitação é imposta pelos sistemas operativos *Symbian*, existentes nos telemóveis disponíveis para desenvolvimento, que determinam a dimensão máxima destes a 512 *bytes*. Não foi no entanto possível verificar se tal limitação existe em dispositivos suportados por outros sistemas operativos mas os testes efectuados em emuladores acedendo à Internet através das mesmas redes móveis, não revelaram tal limitação. Nenhuma documentação foi encontrada que suporte esta teoria mas é de conhecimento geral que a implementação da pilha de protocolo TCP/IP (RFC 1180) pode variar entre sistemas operativos. A título de exemplo, a pilha de protocolo TCP/IP da Microsoft descarta silenciosamente fragmentos em alguns casos: "*Quando um datagrama UDP é maior que o MTU do suporte físico e quando não existe nenhuma entrada ARP referente ao sistema anfitrião destino, a implementação do protocolo TCP/IP da Microsoft mantém apenas o último fragmento do datagrama transferido enquanto espera por uma resposta ARP. O resto dos fragmentos são descartados silenciosamente. Este comportamento ocorre por predefinição e está em conformidade com o RFC referente aos requerimentos dos sistemas anfitriões onde é referido que o ARP deve guardar pelo menos um pacote.*" [7]. A razão apontada para esta limitação fará sentido se considerarmos que o MTU mais comum nas redes móveis celulares é de 512 bytes e caso sejam enviados pacotes de dimensão superior, estes serão fragmentados e eventualmente descartados pelo sistema operativo do telemóvel. Esta limitação referente à dimensão dos datagramas, juntamente com o facto dos *players* nos telemóvel utilizados (Nokia 6600 e 6630) só reproduzirem pacotes de som com um mínimo de 500 ms (4000 *bytes*), implica, ao contrário do TCP, a necessidade de se efectuar *buffering* do som no cliente.

IV.3.2.2 Alterações no lado do cliente

No cliente do telemóvel, depois de estabelecida uma sessão de VNC, o utilizador pode, através do menu de opções, escolher o volume de som que deseja. Um valor superior a '0' iniciará uma sessão de transferência de som com o servidor caso esta ainda não exista e o valor '0' colocará fim a esta. Se o servidor não tiver esta funcionalidade disponível é mostrado um aviso quando se tenta estabelecer a sessão de transferência de som e a aplicação prossegue sem a mesma.

Dado a limitação da maioria dos telemóveis que apenas permitem utilizar protocolos de *streaming* nos seus *players* nativos (p.e: *Real Player*), desenvolvidos sobre o código nativo dos telemóveis, e dado que a API do Java utilizada requer que os *player* carreguem para memória todo o pacote de som a reproduzir antes de o poderem tocar, revelou-se necessário ter dois *players* disponíveis em simultâneo no cliente. O *player* é um mecanismo inerente da MMAPI (*Mobile Media API*) [12] que permite reproduzir áudio e vídeo de forma não bloqueante para a aplicação onde se insere. Desde a sua criação até ser terminado um *player* passa por 5 estados distintos. Quando é criado, encontra-se no estado UNREALIZED. A invocação do método *realize()* faz com que este passe para o estado REALIZED e reúna a informação necessária para poder adquirir o áudio ou vídeo a reproduzir. Ao invocar o método *prefetch()* o *player* passa para o estado PREFETCHED, onde pode ter de adquirir recursos exclusivos, passar para *buffers* internos o áudio ou vídeo a reproduzir e realizar outras tarefas de inicialização que consomem tempo. Assim que o *player* se encontre neste estado, pode começar a reproduzir. A invocação deste método reduz a latência de inicialização do *player* ao mínimo. O início da reprodução é realizado através da invocação do método *start()* que provoca a transição para o estado STARTED. Este método retorna assim que a reprodução tiver início, sendo esta realizada em paralelo com o processo que lhe deu origem. A reprodução terminará automaticamente quando o fim do áudio ou do vídeo for alcançado e o *player* transitará para o estado CLOSED. Neste último estado o *player* já libertou a maioria dos recursos utilizados e não deve ser usado novamente.

Deste modo, se houver áudio para reproduzir, um *player* estará a reproduzi-lo enquanto o outro estará a carregar para memória a próxima amostra de som que está a ser recebida. Esta solução, que pode ser analisada na **Figura IV.11**, é necessária para reduzir o silêncio que um só *player* introduz entre amostras de áudio sucessivas. Actualmente, apenas algumas versões mais recentes das plataformas disponibilizadas pela *Sony Ericsson*, pela *Motorola* e pela terceira edição da série S60 da *Nokia* suportam o protocolo RTP/RTSP (poderão haver outras das quais não tive conhecimento). Só a título de exemplo, o suporte para *streaming* RTSP foi adicionado na série *S60 2nd Edition Feature Pack 3* da Nokia, sendo que o telemóvel disponível para desenvolvimento e testes foi o 6630 que pertence à série *S60 2nd Edition Feature Pack 2* (o N70 já é um dispositivo *S60 2nd Edition Feature Pack 3*).

Deste modo, para não limitar a utilização da aplicação resultante aos telemóveis mais recentes e topo de gama, resolveu-se adaptar a solução de ter dois *players* para reproduzir o efeito de *streaming* e assim manter a aplicação o mais abrangente possível em termos de plataformas alvo.

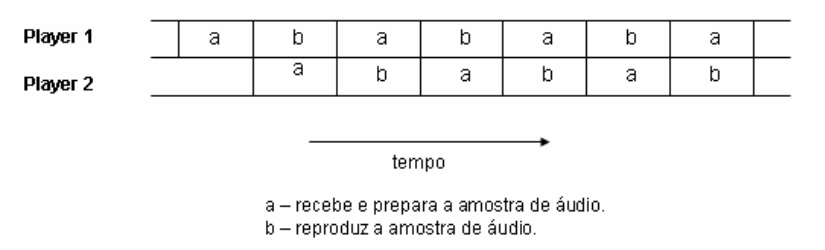


Figura IV.11 – Representação temporal do funcionamento alternado dos dois players.

Mesmo com este sistema de dois *players*, nota-se uma pequena falha entre as amostras de áudio, quase imperceptível durante os testes efectuados no emulador do ambiente de desenvolvimento, mas bastante mais acentuada nos testes efectuados no telemóvel. Isto deve-se ao facto de no emulador, o método *realize* do *player* apenas retirar a informação relativa ao cabeçalho do áudio a reproduzir e quando o método *start()* do *player* é invocado, são realizadas leituras sucessivas da fonte de áudio com *buffer* relativamente curto (< 1 *Kbyte*), o que torna quase instantânea e imperceptível a troca entre *players*. No entanto, dos telemóveis Nokia disponíveis para testes nenhum apresentou um comportamento semelhante, por utilizarem *buffers* maiores. Apenas o telemóvel HTC TyTn se aproximou do comportamento obtido nos emuladores. Apesar deste facto, esta falha no áudio é bastante inferior à encontrada numa solução com um só *player*. Por exemplo, no caso do telemóvel Nokia 6630, um *player* leva entre 300 a 400 ms a preparar uma amostra de som para reproduzir (**Secção V.5.1**) e mais um tempo indeterminado, poucas centenas de milissegundos, até inicializar realmente a sua reprodução. Isto significa que a falha existente no som, de poucas centenas de milissegundos no caso dos dois *players*, passará para cerca de 500 ms ou superior, no caso de um só *player*. No dispositivo HTC TyTn esta diferença não é tão acentuada pois o tempo de preparação do *player* é inferior a 20 ms e o tempo que este leva a inicializar a reprodução da amostra de áudio, apesar de indeterminada, é também na ordem das poucas dezenas de milissegundos, tornando-se por vezes imperceptível.

Para reduzir a percepção do silêncio entre amostras, utilizam-se pacotes de som de 4 segundos, pois é menos perceptível um silêncio de poucos milissegundos de 4 em 4 segundos, do que esse mesmo silêncio de 2 em 2 segundos ou de 1 em 1 segundo. Apesar de amostras de dimensão superior reduzirem ainda mais este efeito, apresentam no entanto contrariedades. Nomeadamente, aumentam a diferença entre o instante de tempo em que o som ocorre no servidor e o instante em que este é reproduzido no cliente (*lag*), sendo a recepção, descodificação dos pacotes de som e a preparação dos *players* também mais demorada.

IV.3.2.2.1 Diferenças entre a solução em TCP e a solução em UDP

Na **solução em TCP**, todo o código que implementa o som encontra-se numa só *thread*, sendo que esta fica inicialmente à espera de receber 44 bytes de cabeçalho WAVE do servidor, para poder criar os *buffer's* necessários ao seu correcto funcionamento. Seguidamente, a *thread* entra num ciclo

composto por 4 etapas que podem ser interpretados no diagrama de blocos apresentado na **Figura IV.12**:

- Recebe pacote de som e descomprime-o se estiver comprimido.
- Prepara *player* 1 e reproduz amostra de som se o *player* 2 já tiver terminado.
- Recebe pacote de som e descomprime-o se estiver comprimido.
- Prepara *player* 2 e reproduz amostra de som se o *player* 1 já tiver terminado.

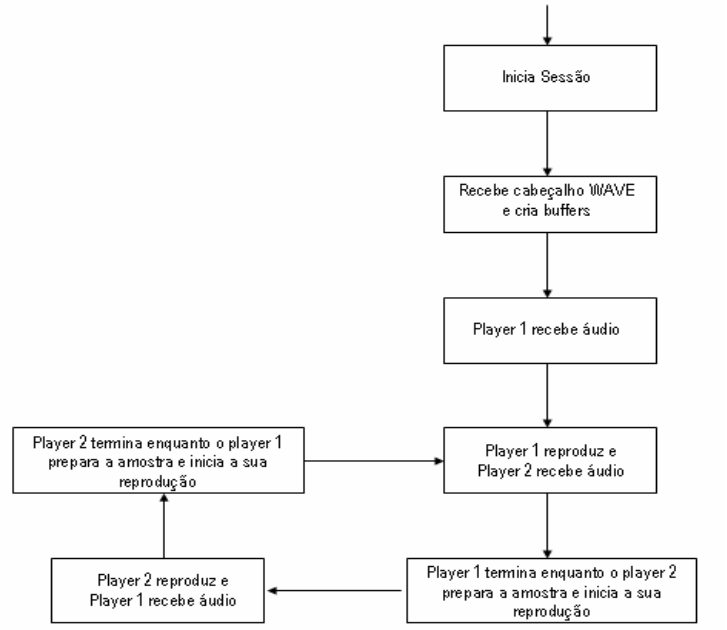


Figura IV.12 – Diagrama de blocos da solução em TCP para a transferência de áudio.

Antes de enviar cada pacote de som, o servidor envia 4 bytes contendo o tamanho deste. O cliente utiliza esta informação para saber a quantidade de informação que compõe o próximo pacote, isto porque o TCP, enviando os dados numa sequência de bytes (*stream*), não delimita os pacotes.

Este sistema têm o inconveniente de estar à mercê das variações da rede, ou seja, se a rede durante um intervalo de tempo tiver uma quebra de largura de banda abaixo dos 64kbps (requisito necessário para transferir 1 segundo de som em 1 segundo de tempo, no formato utilizado) ou um pouco menos dependendo da compressão do pacote, este não será recebido na totalidade durante a reprodução do pacote anterior. Por consequente, o silêncio entre pacotes será acrescido do tempo que leva a terminar de receber o novo pacote para além da duração da reprodução do pacote anterior, mais o tempo de descodificação e preparação do *player*. A variação do débito da rede, relativa a testes efectuados à transferência de som, pode ser observado na **Secção V.5.1**.

Na **solução em UDP**, o código que trata da implementação do som divide-se em duas *threads* e pode ser interpretado no diagrama de blocos apresentado na **Figura IV.13**:

- A *thread* de recepção, que se dedica a receber os pacotes de som, a descomprimi-los se vierem comprimidos e a armazená-los em memória (*buffering*).
- A *thread* de gestão do áudio, que inicialmente envia uma mensagem para o servidor pedindo o início de uma sessão de transferência de som. Posteriormente, recebe o

cabeçalho WAVE enviado pelo servidor que para além de servir para dimensionar o *buffer* de recepção, serve também de confirmação de que a mensagem de pedido de início de sessão foi transmitida com sucesso. De seguida cria a *thread* de recepção e gere o áudio armazenado por esta, distribuindo-o pelos dois *players*. Isto é, sempre que o áudio armazenado corresponder a mais de 4 segundos de som, um *player* é preparado para reproduzi-lo e assim que o outro *player* deixe de tocar (caso o esteja a fazer), este inicia a reprodução da próxima amostra de áudio.

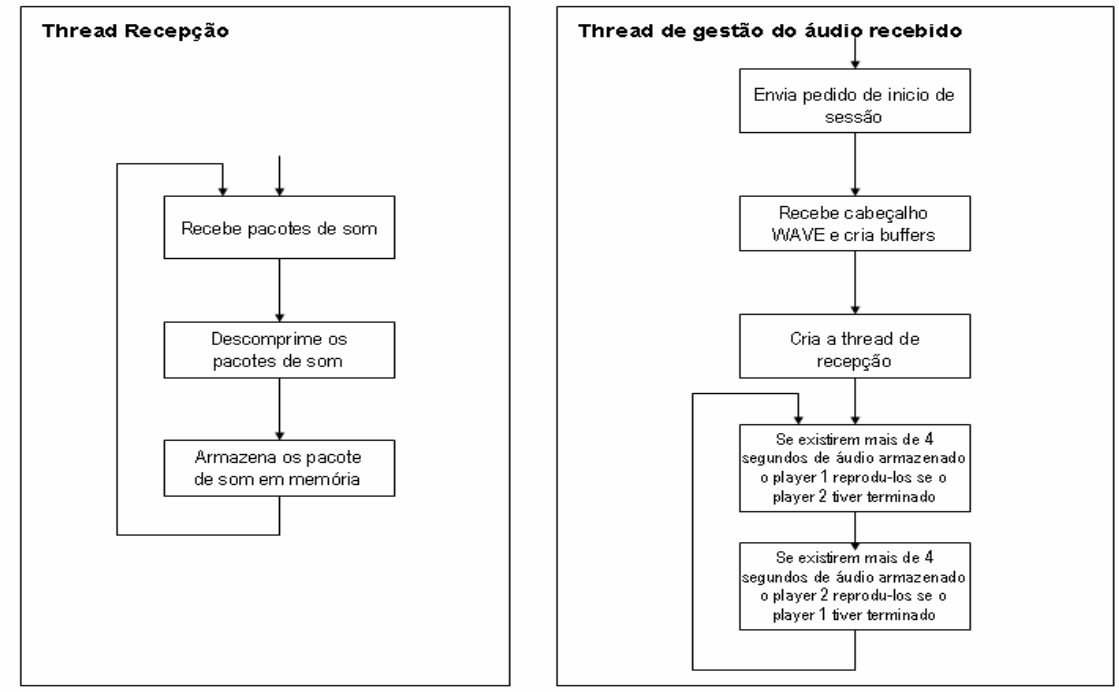


Figura IV.13 – Diagrama de blocos da solução em UDP para a transferência de áudio.

Foi escolhido um tamanho de *buffering* de 16 segundos, correspondente a 4 amostras de 4 segundos, para permitir que esteja sempre áudio disponível para um *player* carregar enquanto o outro está a tocar, tendo sido, entre a alternativa de 8 segundos, o que apresentou melhores resultados.

Esta solução, apesar de minimizar o inconveniente de uma possível redução temporária da largura de banda disponível, encontrado na solução por TCP, introduz no entanto outras situações inoportunas. Por exemplo, se tiver a ser reproduzida uma música no servidor, se a largura de banda for suficiente para permitir encher o *buffer* do cliente, e se de um momento para o outro o cliente resolver mudar a música ou parar a sua reprodução no servidor, existirão 16 segundos de intervalo entre a acção do cliente e a reacção ao nível do som audível. Outro inconveniente é que o sistema de supressão de silêncio deixa de funcionar como inicialmente previsto, isto é, funciona mas para além de suprimir o silêncio, suprime também o tempo que devia de se "ouvir" esse silêncio, o que pode não ser desejável. Por exemplo, se uma música contiver um instante de silêncio a meio, o servidor não enviará os pacotes de áudio correspondentes a este silêncio para o cliente mas enviará o restante som. Como o cliente acumula no *buffer* apenas os pacotes recebidos, este irá juntar o último som ao

novo, eliminando o instante de silêncio no existente no meio, parecendo ao utilizador que a música não teve pausa nenhuma.

Para uma melhor compreensão do que foi descrito anteriormente, apresenta-se na **Figura IV.14** os diagramas de mensagens entre cliente e servidor, referentes à transferência de som tanto em TCP como em UDP.

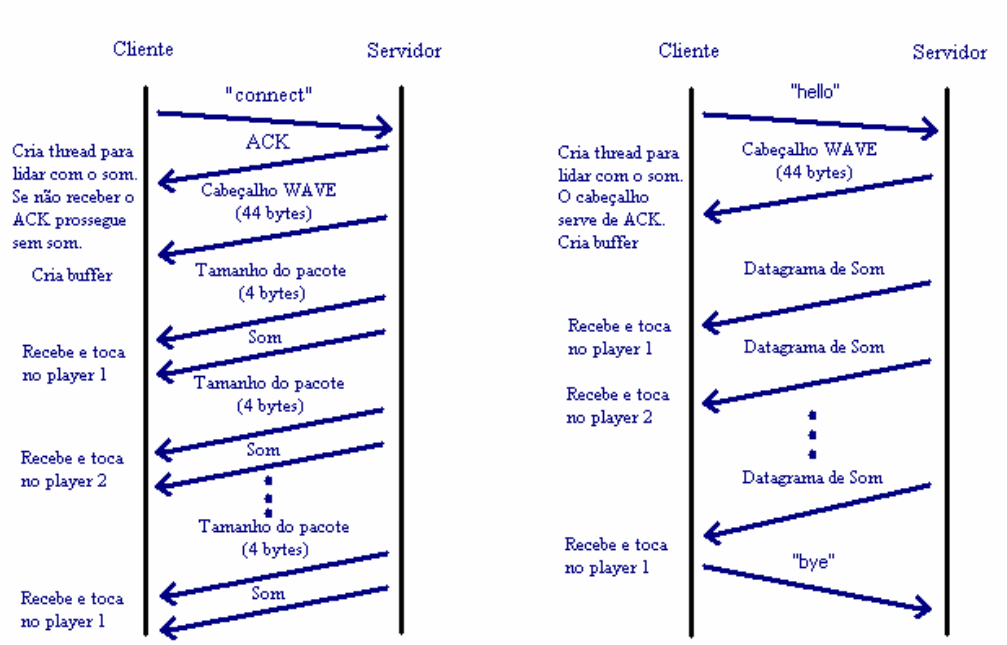


Figura IV.14 – Diagrama de mensagens da solução em TCP (lado esquerdo) e da solução em UDP (lado direito)

IV.3.3 Compressão de som e supressão de silêncios

A compressão do áudio no lado do servidor baseia-se no algoritmo simples de suprimir símbolos repetidos, ou seja, se no pacote de som se encontrar uma sequência de *bytes* "000000000000", esta será comprimida para a sequência de bytes "< 0 12" onde o símbolo '<' (*byte*) indica o início de uma sequência de símbolos que foi comprimida, o '0' (*byte*) indica o símbolo repetido, e o '12' (*short* – 2 *bytes*) indica o número de repetições deste.

Esta compressão só é efectuada para repetições de 5 ou mais símbolos/*bytes*, pois a codificação de menos símbolos numa notação de 4 *bytes* não traz ganhos, antes pelo contrário, exige mais tempo para ser descodificado no cliente.

No caso de ser encontrado o símbolo '<' no meio do pacote de som, este será sempre codificado no novo formato mesmo que não apareça repetido. Deste modo o símbolo '<' passará a "< < 01", existindo uma perda de compressão, necessária para manter a coerência e para ser possível descodificar correctamente a amostra de áudio no cliente.

Os algoritmos (pseudo código) utilizados para compressão e descompressão dos pacotes de áudio podem ser consultados no **Anexo C**. A escolha do símbolo '<' foi realizada por análise da sequência de *bytes* que compõem alguns pacotes de som, tendo sido escolhido aleatoriamente de entre os símbolos que aparentavam repetir-se menos. Para tal, foram armazenadas em ficheiros

várias amostras de som relativamente a silêncio, captura de áudio através do microfone e diferentes músicas a serem reproduzidas no computador, tendo sido posteriormente criado um *script* para percorrer estes ficheiros e fazer a contagem individual das aparências de cada símbolo. Um exemplo dos ficheiros de resultados obtidos pode ser consultados no **Anexo D**.

A supressão de silêncio, é um método que basicamente se aproveita do facto dos pacotes de áudio sofrerem uma compressão maior quando se trata de momentos de silêncio. Deste modo, pode-se definir um limiar a partir do qual os pacotes, depois de comprimidos, são interpretados como ausência de som ou não. Este limiar é definido pelo utilizador para um maior ajuste à máquina em questão. Por exemplo, um pacote de 32000 *bytes* de áudio representando "silêncio" pode ser facilmente comprimido para menos de 10000 *bytes* com este processo. Isto depende de ser áudio interno que vai para a placa de som ou de ser áudio capturado pelo microfone, pois neste último caso, o ruído de fundo sempre existente, não deixa a compressão atingir os valores referidos. Se tratar-se de um pacote todo ele com música com bastante ritmo, a compressão pode ser nula ou de apenas algumas centenas ou poucos milhares de *bytes*.

Deste modo, se definirmos no servidor o limiar de silêncio, por exemplo, nos 10000 *bytes*, todas as amostras de áudio que depois de comprimidas tiverem uma dimensão inferior, serão consideradas representativas de momentos de silêncio e não serão enviados ao cliente. Alguns valores para a compressão dos pacotes de áudio podem ser observados na **Figura V.5**.

IV.4 Perfis de Utilizador

Tendo em consideração que a interface do telemóvel é ainda relativamente limitada comparada com a interface do PC (teclado + rato) e que por isso o controlo deste último através do telemóvel requer alguma habilidade e paciência por parte do utilizador, uma nova funcionalidade implementada foi, um perfil de utilizadores do lado do servidor. A ideia é permitir a criação de novos tipos de utilizadores, concedendo ou não a estes, o controlo remoto do rato e/ou do teclado, a escolha ou não, de uma aplicação específica para o servidor disponibilizar ao cliente (o ambiente de trabalho visível no telemóvel fica limitado à área da aplicação seleccionada) e a possibilidade de para certos utilizadores, desactivar ou não o *Wallpaper*, padrões de fundo e efeitos do Windows, que são recursos que consomem largura de banda. A ideia é também a de permitir diferenciar clientes de telemóvel, onde a largura de banda é um factor importante, de clientes a aceder através de um computador pessoal numa rede local ou mesmo pela Internet, onde a largura de banda já não é um factor tão crucial. Claro que para tal ser possível é necessário alterar os clientes existentes para PC's para suportarem a extensão ao protocolo utilizada.

Deste modo, o utilizador não necessita de aceder a todo o ambiente de trabalho e lançar as aplicações desejadas, bastando para tal, que seja efectuada uma simples configuração prévia no servidor. Para melhor se descrever esta situação, apresenta-se na **Figura IV.15**, um cenário em que um utilizador acedeu ao servidor para ter acesso somente à linha de comandos e um outro caso em que acede somente para ter acesso ao jogo *Minesweeper* do Windows. Não representado nas

imagens está o facto de num dos casos o utilizador apenas poder controlar o teclado remoto e no outro apenas poder controlar o rato.

Figura IV.15 – Exemplo de perfis de utilizador para acesso a aplicações distintas.

Para o servidor saber qual o utilizador que está a aceder existiam duas possibilidades: ou se concebia uma nova mensagem para o protocolo RFB como se fez para as alterações ao factor de escala do lado do servidor, ou se introduzia uma extensão ao protocolo inserindo um novo tipo de segurança (**Secção A.5.1.2 e Secção A.5.2**).

A primeira opção tinha o aspecto negativo de fazer com que a sessão VNC fosse terminada abruptamente caso o cliente enviasse essa nova mensagem para um servidor que não a conhecesse. Para além disso, esta mensagem só poderia ser trocada depois de estabelecida a sessão e portanto era tempo desaproveitado até o cliente ter conhecimento de que o servidor em questão não suporta a extensão de perfis de utilizador.

A segunda opção, e a que foi implementada, tem a característica de apenas ter de se criar um novo tipo de segurança do qual o cliente e o servidor tenham conhecimento, para assim se fazer a verificação do utilizador. Como na versão 3.3 do protocolo RFB, que é a versão utilizada neste projecto, quem decide o tipo de segurança a utilizar é o servidor, foi necessário acrescentar mais um parâmetro nas configurações deste, para dar a possibilidade de se escolher o tipo de autenticação a utilizar. No caso de se optar pela autenticação normal, permite-se que clientes já existentes se liguem, ou mesmo o cliente de telemóvel, sendo neste caso ignorado o campo de utilizador. No caso de ser utilizada a autenticação que permita a utilização dos perfis de utilizador, o servidor fica à espera de receber do cliente um nome de utilizador, e alterará as suas propriedades consoante o que estiver definido no perfil associado.

Deste modo, caso seja escolhido este último tipo de autenticação, o servidor, durante a troca de mensagens iniciais e na fase da escolha do tipo de segurança a utilizar, enviará ao cliente a mensagem representada na **Tabela IV.3**:

Tabela IV.3 – Formato da mensagem indicando o tipo de segurança a utilizar.

Numero de bytes	Tipo	[Valor]	Descrição
4	U32	FFFF	*tipo de segurança*
16	U8		*desafio*

Esta mensagem, contendo um desafio para confirmar a palavra-chave a ser utilizada pelo cliente, faz com que este último responda, enviando não só a resposta ao desafio, como acontece no modo de autenticação normal, mas também o nome do utilizador associado ao perfil pretendido, como se pode ver na mensagem representada na **Tabela IV.4**:

Tabela IV.4 – Formato da mensagem contendo o nome de utilizador e a resposta ao desafio.

Numero de bytes	Tipo [Valor]	Descrição
16	U8	*nome do utilizador*
16	U8	*resposta ao desafio*

Para conseguir este objectivo, foi gerada uma estrutura de dados no servidor, de fácil configuração através de uma interface gráfica (representada na **Figura IV.16**), que armazena as propriedades dos diferentes perfis de utilizador. O formato utilizado para a estrutura em questão pode ser visualizado na **Figura IV.17**.

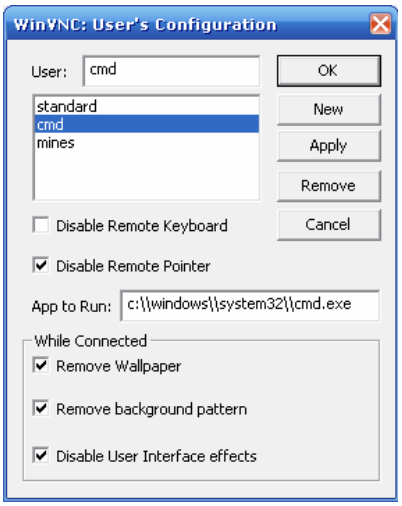


Figura IV.16 – Interface gráfico para configuração dos perfis de utilizador.

```
typedef struct _users{
    char user[16];
    BOOL wallpaperDisabled;
    BOOL patternsDisabled;
    BOOL effectsDisabled;
    char App[200];
    BOOL keyboardDisabled;
    BOOL mouseDisabled;
} UserProfile;
```

Figura IV.17 – Estrutura de dados utilizada para armazenar os perfis de utilizador.

Esta informação é armazenada num ficheiro de texto, num formato simples que se definiu. Resolveu-se não a guardar no registo do Windows como acontece com as outras propriedades do VNC, pois para além desta informação não ter um tamanho fixo, podem ser constantemente apagados e criados novos utilizadores, o que tornaria a manipulação do registo e a gestão do seu conteúdo mais delicada e cuidada. Quando se inicia a aplicação do servidor, este verifica se o ficheiro

contendo esta informação existe e se esta se encontra no formato adequado. Caso isto não se verifique, é criado um novo ficheiro contendo um utilizador base (*standard*), com algumas propriedades predefinidas, que podem depois ser alteradas. Caso o ficheiro exista e contenha o formato correcto, as definições dos utilizadores existentes no ficheiro são carregadas para a memória na estrutura apresentada. Alterações, criação de novos utilizadores ou eliminação de utilizadores existentes são automaticamente gravadas para o ficheiro, para manter uma coerência entre a informação existente em memória e a informação armazenada em disco para futuras utilizações.

No lado do cliente, foi criada uma caixa de texto que permite introduzir o nome de utilizador que se pretende utilizar. No caso deste utilizador não existir (não estar configurado no servidor), este campo não ser preenchido no telemóvel ou de o servidor estar em modo de autenticação normal, serão utilizadas as propriedades definidas no utilizador base (*standard*), que não pode ser apagado do servidor mas apenas alterado.

Se um cliente “normal”, ou seja, um cliente que não contém a extensão para este novo tipo de segurança, se tentar ligar ao servidor quando este está a exigir autenticação para a utilização dos perfis de utilizador, será apresentado um erro de segurança e a sessão não será estabelecida.

Capítulo V. Ferramentas e testes de desempenho

V.1 Introdução

Sendo outro dos objectivos do trabalho o estudo das capacidades e características das redes móveis celulares (GPRS/UMTS) e da consequente viabilidade do projecto a operar sobre as mesmas, foram realizados alguns testes com o intuito de estimar alguns parâmetros específicos destas redes.

Para este efeito, existe um registo (*Log*) onde são armazenados os valores referentes a cada transferência de dados/som efectuada, existe um formulário de estatísticas (*stats*) onde se podem visualizar os valores totais, mínimos, médios e máximos relativos às transferências de dados/som anteriores e existe um formulário onde se pode observar o decorrer de um *Ping* ao servidor e os correspondentes tempo de ida e volta obtidos.

Um dos parâmetros que não é aqui calculado é a variação de atraso na rede (*jitter*) porque implica uma sincronização dos relógios entre o cliente e o servidor que não foi possível obter. A obtenção deste valor implicaria que os pacotes de som enviados pelo servidor fossem marcados com o instante de envio (com um *timestamp*) e que de tempos a tempos fossem trocadas mensagens entre o servidor e o cliente para sincronizar os relógios respectivos.

V.2 Registo (*Log*)

Na aplicação original, para a qual se resolveu dar o contributo, este registo guardava os vários passos da troca de mensagens inicial entre o cliente e o servidor, as teclas pressionadas pelo utilizador, os pedidos de actualizações realizados ao servidor e correspondentes respostas detalhadas, etc.

Após a aplicação estar testada e aparentemente sem *bugs*, resolveu-se utilizar o registo para apresentar valores de parâmetros à medida que iam surgindo, mantendo deste modo, um registo não só de valores médios (no formulário de estatisticas) mas também de valores “instantâneos”. Manteve-se no entanto, o registo da troca de mensagens iniciais com o servidor, para permitir perceber porquê que eventualmente uma sessão falhou e regista-se ainda alguma informação adicional para permitir o *debug* da aplicação.

Os valores calculados e apresentados neste registo são divididos por transferência de dados (mensagens RFB referentes a actualizações do ambiente de trabalho) e por transferência de som. Para cada transferência de dados apresenta-se, para além do instante de tempo em que ocorre (em milissegundos desde o início da sessão VNC), a dimensão em *bytes* da actualização, o tempo que esta demorou a ser recebida por inteiro, e com estes dois valores calcula-se uma estimativa do valor médio do débito *downstream* da rede durante este intervalo de tempo. O valor do débito da rede, neste caso, vem afectado pelo tempo de processamento dos dados, à medida que estes vão sendo recebidos, e pelo correspondente desenhar no ecrã, por parte do telemóvel. Deste modo, este valor não pode ser considerado uma estimativa exacta mas serve para dar uma ideia deste parâmetro, até

porque realizando as alterações ao factor de escala no lado do servidor, onde o processamento efectuado pelo telemóvel é inferior, as estimativas são razoavelmente idênticas.

No caso da transferência de um pacote de som, são apresentados para além dos valores referidos no caso anterior, o tempo que a amostra leva a ser descodificada e o tempo que se levou a preparar o *player* para reproduzir essa mesma amostra. O valor aqui obtido para o débito da rede pode ser considerado mais exacto que o anterior, pois o pacote é recebido por inteiro e só posteriormente é efectuado o seu processamento.

Seleccionaram-se estes métodos para cálculo do débito da rede porque utilizam as transferências de informação nativas à aplicação, não obrigando a tráfego adicional, e porque para actualizações ao ambiente de trabalho completo e nas transferências de som (em TCP), a quantidade de informação transferida é na ordem das dezenas de *kilobytes*, o que permite obter estimativas com alguma precisão.

Durante uma actualização, ou mesmo durante uma transferência de som, apenas são transferidos dados do servidor para o cliente, podendo-se através de uma simples operação de divisão, calcular o débito de *downstream* da rede, sendo que este é representado pelo número de bits transferidos por unidade de tempo (*kbps*).

Quando se está a transferir som em paralelo com uma actualização, a estimativa do débito já não é real pois parte da largura de banda está a ser utilizada pela transferência de som e a outra parte pelos dados referentes à actualização. Tendo isto em consideração, o cálculo mais exacto do débito da rede, obtém-se quando apenas se transfere áudio durante um certo intervalo de tempo sem proceder a actualizações nenhumas durante esse mesmo período.

Como a quantidade de dados transferidos do cliente para o servidor, *upstream*, é bastante reduzida, achou-se irrelevante estar a calcular o débito nesta direcção. Na **Figura V.1** pode ser visualizado um exemplo de um registo, capturado no início de uma sessão, onde apenas foi realizada uma actualização ao ambiente de trabalho e transferido apenas um pacote de áudio.

Figura V.1[1] **–** Exemplo de um registo capturado no início de uma sessão.

[1] Foi realizada uma colagem de várias partes da imagem para poder concentrar o registo numa só figura.

Para permitir o fácil acesso a esta informação, foi introduzida a possibilidade deste registo ser transferido por uma ligação TCP para o servidor. Por sua vez, o servidor a correr a aplicação *Netcat,* ou outra similar, envia toda a informação recebida para um ficheiro. Posteriormente, com o auxilio de um *script*, elaborado para o efeito, converte-se a informação relevante para um ficheiro ".csv" (comma-separated values) que pode então ser interpretado por um editor de folha de cálculo como o Excel.

V.3 Estatísticas (*Stats*)

Neste formulário, pode ser visualizado o último valor calculado para o débito da rede, bem como o valor mínimo, médio e máximo deste. Indica-se a quantidade de *bytes* recebidos pela aplicação referentes aos dados (mensagens do protocolo RFB), referentes ao áudio e à soma total dos dois e apresenta-se também o número de actualizações efectuadas e os valores mínimos, médios e máximos da dimensão e duração das mesmas. É ainda possível ao utilizador verificar a memória utilizada pela aplicação e a duração da ligação ao servidor, o que pode ser útil para redes GPRS onde alguns operadores fornecem tarifários com taxação por unidades de tempo.

Quando se encontra estabelecida uma sessão de transferência de som, pode ainda ser visualizado neste formulário, os valores mínimos, médios e máximos do tempo de recepção dos pacotes de áudio, do tempo que levam a ser descomprimidos e do tempo que os *players* levam a se preparar para os reproduzir. Como os pacotes de som podem vir comprimidos com dimensões consideravelmente discrepantes, para permitir o cálculo de valores médios com algum significado, estes foram agrupados em 3 segmentos de igual dimensão. Os valores utilizados para efectuar esta divisão são relativos à dimensão do *buffer* de som (valor retirado do cabeçalho WAVE que é enviado inicialmente pelo servidor). Na **Figura V.2** pode ser visto um exemplo de um formulário contendo as estatísticas de uma sessão.

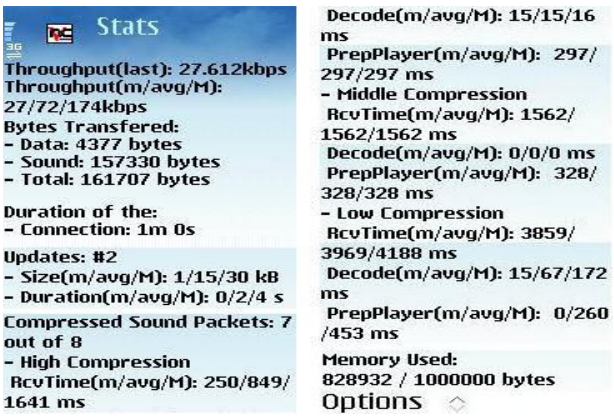


Figura V.2[2] **–** Exemplo das estatísticas de uma sessão.

[2] Foi realizada uma colagem de várias partes da imagem para poder concentrar as estatísticas numa só figura.

V.4 Ping

Dado que o J2ME não suporta pacotes ICMP's como sendo o de *ping*, as alternativas para o cálculo do tempo de ida e volta (RTT) da ligação são:

1. Utilizar um servidor de eco sobre UDP e calcular o tempo entre o instante em que o pacote é enviado e o instante em que este é depois recebido. Para este efeito, o cliente envia um pacote UDP contendo um número de sequência e o instante em que está a ser colocado na rede. No lado do servidor está implementado uma *thread* que se limita a estar à escuta num porto predefinido e a reenviar para a origem os pacotes recebidos (faz eco dos pacotes). Quando um pacote chega ao cliente, este verifica o seu número de sequência, para averiguar se não houve reordenação dos pacotes. Seguidamente, analisa o instante de tempo nele contido e compara-o com o instante de tempo actual, obtendo assim uma estimativa do tempo de ida e volta.

   Resolveu-se não utilizar o porto normalmente associado ao servidor de eco (porta 7), pois este serviço foi retirado dos sistemas operativos mais recentes por ser facilmente alvo de ataques de DoS (*denial of service*). Deste modo, se a porta utilizada pelo VNC for a 5900, a 5901 estará reservada para estabelecer sessões de transferência de som e a 5902 estará associada ao servidor de eco.

2. Utilizar um servidor de eco sobre TCP, que ao contrário dos de UDP, recebem os segmentos em *buffers*, fazem a sua reconstrução, enviam uma confirmação à origem e só depois fazem o eco da mensagem. Existe assim um atraso de reconstrução aleatório entre o instante em que a mensagem é recebida no servidor de eco e o instante em que é verdadeiramente realizado o seu eco. Este facto poderia ser considerado como uma estimativa do tempo que o cliente e servidor demoram efectivamente a trocar segmentos, considerando que toda a aplicação funciona sobre *sockets* TCP e por consequente sofre deste atraso. No entanto, esta alternativa não se revelou simples de implementar pois quando se efectua uma ligação a um *socket* TCP num porto conhecido, é gerado um novo *socket* num porto aleatório, devolvido pela chamada de sistema *accept*, e a comunicação efectua-se através deste. Como a maioria dos PC's a que se pretende aceder encontram-se por detrás de um comutador, é necessário configurar neste os portos utilizados para comunicação e redireccioná-los para o PC (NAT). Como o pedido de ligação é efectuado num porto conhecido mas a comunicação em si é efectuada através do *socket* criado especificamente pelo sistema operativo, num porto aleatório, para a sessão TCP em questão, não é possivel configurar o NAT do comutador e assim enviar dados do cliente (telemóvel numa rede externa) para o servidor (por detrás de um comutador numa rede local). Deste modo, optou-se por não se implementar esta solução.

3. Outra solução encontrada e implementada foi a de estimar o tempo de ida e volta através dos cabeçalhos TCP trocados durante o início de uma sessão. Para tal, o cliente envia um pacote TCP SYN, para um porto predefinido no servidor que se espera não estar a ser utilizado e o servidor responde com um pacote TCP RST (o porto apenas é predefinido para se poder

configurar a tabela NAT no comutador). Utiliza-se o facto de não ser estabelecida nenhuma sessão TCP para evitar o efeito da janela de congestão que pode resultar num estado TCP TIME_WAIT. Dadas as limitações do J2ME associado ao MIDP1.0/CLDC1.0 utilizado, não se consegue detectar o erro no *socket* EONREFUSED, que é gerado pelo pacote TCP RST. No entanto, é gerada uma excepção ao não se conseguir estabelecer a ligação e quando tal acontece, calcula-se a diferença entre o instante actual e o instante em que se enviou o pedido de ligação. Pode-se observar na **Figura V.3** a troca de mensagens efectuada entre cliente e servidor, durante uma tentativa de início de sessão falhada e como esta é utilizada para o cálculo do tempo de ida e volta.

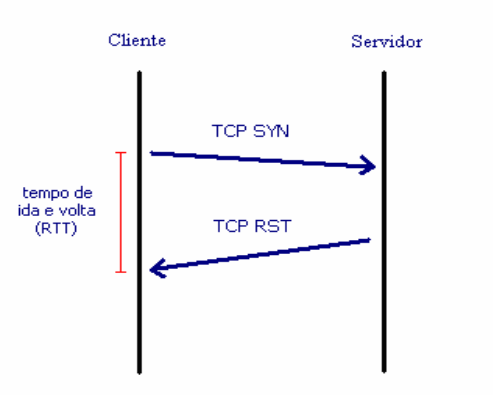


Figura V.3 – Troca de mensagens durante um início de sessão TCP falhado.

Os métodos implementados não permitem no entanto, calcular o número de pacotes perdidos porque não é possível alterar a duração do temporizador associado aos *sockets* em J2ME. Assim se um pacote é perdido, o J2ME só gera uma excepção ao fim de 150 segundos, não sendo agradável ao utilizador ter de esperar tanto tempo por cada pacote perdido.

Dado que o tempo de ida e volta varia durante a sessão, não se envia apenas um pacote mas sim 5, para se poder fazer uma média e apresentar ao utilizador um valor mínimo, médio e máximo.

O método para cálculo do tempo de ida e volta sobre UDP foi utilizado para comparar com os tempos obtidos pelo método de TCP implementado. Na **Figura V.4** pode se observar um exemplo de um formulário contendo um *Ping*.

Os resultados obtidos para o tempo de ida e volta, tanto pelo método sobre UDP, como pelo método sobre TCP, servem também para nos dar uma indicação de que a variação de atraso na rede é uma realidade. No entanto pouca informação poder ser retirada destes valores pois o número de amostras não é considerável.

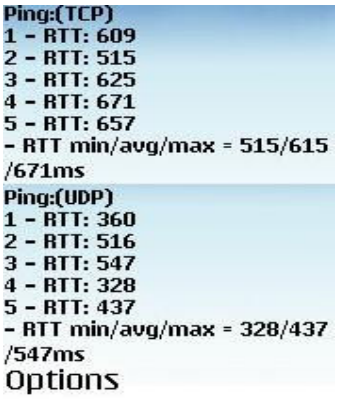


Figura V.4[3] **–** Exemplo de um Ping efectuado durante uma sessão VNC.

V.5 Testes

V.5.1 Transferência de Som

Para comparar os dois métodos de transferência de som foram realizados alguns testes e recolhidos dados para analisar, tendo estes sido resumidos na **Figura V.5** e na **Figura V.6**. Em ambos os casos realizou-se a transferência de som durante cerca de um minuto, enquanto se reproduzia música no servidor. Estes testes serviram também para verificar o débito obtido numa rede UMTS durante o período de execução dos testes, utilizando para tal, um cartão de dados com contracto com a operadora para velocidades até 384 kbps (*downstream*). Posteriormente foram realizados testes também sobre a rede HSDPA com velocidades até 3,6 Mbps (*downstream*).

No caso dos gráficos apresentados para a versão em TCP, podem-se observar os resultados de dois testes realizados em instantes de tempo diferentes (a azul e a vermelho) utilizando o telemóvel Nokia 6630 e de dois testes efectuados com o dispositivo HTC TyTN sobre uma ligação HSDPA (a verde e preto). No caso do UDP, apesar de também terem sido efectuados diversos testes sobre UMTS, resolveu-se não sobrepor os gráficos, que se revelaram semelhantes, pois só ia complicar a observação dos mesmos. Deste modo, é apresentado apenas o resultado de uma experiência efectuada.

V.5.1.1 TCP

A partir da **Figura V.5** facilmente se verifica que o valor médio obtido para o débito durante a execução destes testes foi de 119 kbps para a rede UMTS e de 593 kbps sobre HSDPA. Dado que estes valores são praticamente constante nestes testes (tirando algumas oscilações observadas nos gráficos referentes aos testes efectuados sobre HSDPA), é de esperar que o tempo de recepção dos pacotes seja proporcional ao tamanho dos mesmos, o que de facto se verifica. Ou seja, quando se recebe um pacote de dimensões menores, o tempo que se leva a recebê-lo é também menor e vice-

[3] Foi realizada uma colagem de várias partes da imagem para poder concentrar o formulário de Ping numa só figura.

versa. O desvio padrão obtido para o débito na rede UMTS foi de 15,36 kbps e o intervalo de confiança a 95% é [114,0; 123,4]. Para a rede HSDPA obteve-se um desvio padrão de 94,0 kbps e o intervalo de confiança a 95% é [542,4; 604,7].

No gráfico a azul pode-se também observar, que o pacote número 15, representando momentos de silêncio entre duas faixas de música, foi comprimido de 32000 *bytes* para 5954 *bytes*. Neste caso, se estivesse a ser utilizada supressão de silêncio com, por exemplo, um limiar de 10000 *bytes*, este pacote não teria sido enviado pelo servidor e no cliente nada seria reproduzido. Como a supressão de silêncio não estava a ser utilizada, tornou-se audível o ruído de fundo característico do "silêncio" electrónico, durante os 4 segundos de duração da amostra de som.

Por análise dos gráficos, verifica-se que o tempo de descompressão de um pacote não depende do seu grau de compressão. Seria de esperar que quanto maior fosse a compressão da amostra recebida, maior seria o tempo que a aplicação demoraria a descomprimi-la. No entanto, isto não se verifica. Nos testes realizados utilizando o Nokia 6630, tirando o primeiro pacote, todos os outros levam cerca de 15 ms a serem descomprimidos. Utilizando o HTC TyTn, existe uma maior variação dos tempos de descodificação mas que em nada estão relacionados com a compressão efectuada pois os pacotes recebidos têm aproximadamente todos 32000 bytes. No lado do servidor, a compressão dos pacotes de som leva menos de 10 ms, tendo este valor sido obtido por *debug* à aplicação. Já a análise do último gráfico indica-nos que o tempo que um *player* leva a preparar um pacote de 4 segundos de som para o reproduzir (32000 bytes), é de cerca de 300 a 400 ms no caso do telemóvel Nokia 6630 e de menos de 20 ms para o HTC TyTn. Este tempo no entanto, não se revelou proporcional ao tamanho dos pacotes, mantendo-se aproximadamente constante para pacotes de dimensões inferiores.

Como era desejado, o tempo total que um pacote leva a ser recebido, descodificado e preparado pelo *player* (cerca de 2,5 segundos no Nokia 6630 sobre UMTS e cerca de 600 ms no HTC TyTn sobre HSDPA), é menor do que a duração em termos de áudio de um destes pacotes (4 segundos). Deste modo, a ideia de ter um *player* a reproduzir um pacote de som enquanto o próximo pacote é recebido e preparado para tocar, foi conseguida. No entanto, não se conseguiu eliminar a pequena pausa existente durante a alternância entre os *players* e assim obter um verdadeiro fluxo de som sem interrupções. Ainda assim, obteve-se uma melhor performance utilizando o HTC TyTn onde as pausas entre amostras eram quase imperceptíveis e nem sempre ocorriam, devendo-se tal facto ao menor tempo dispendido pela inicialização dos *players* neste dispositivo. As únicas soluções encontradas para solucionar este problema passam ou por codificar os *players* na linguagem nativa do telemóvel em questão ou por desenvolver esta funcionalidade para um modelo de telemóvel que já suporte *streaming* RTSP, na esperança de que todos os modelos futuros a suportem.

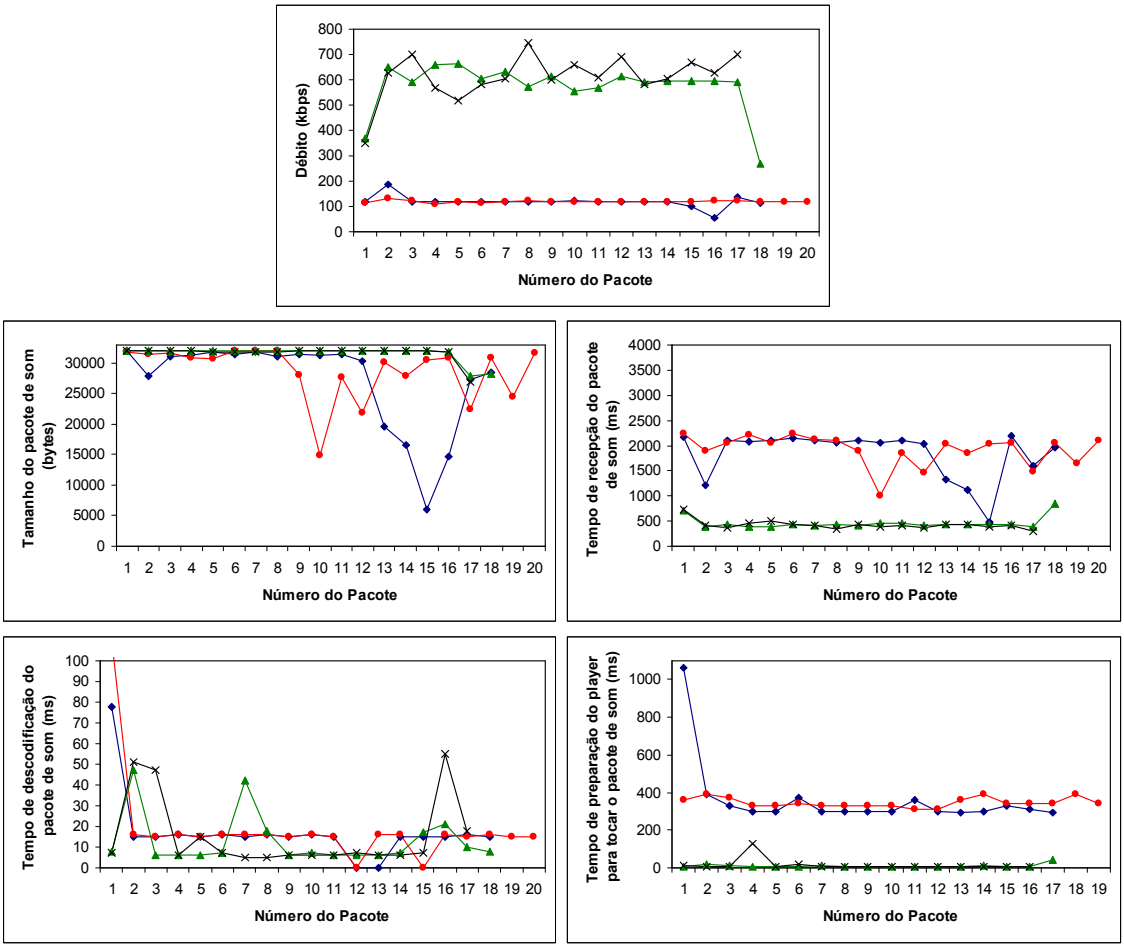


Figura V.5 – Gráficos relativos à transferência de som por TCP durante cerca de 1 minuto (duas sessões distintas sobre UMTS e duas mais sobre HSDPA).

V.5.1.2 UDP

Na **Figura V.6**, comparando com a transferência de som por TCP, rapidamente se verifica que para aproximadamente o mesmo intervalo de tempo, são trocados bastantes mais pacotes de som. Isto deve-se ao facto do tamanho máximo de um datagrama UDP ser de 512 bytes e assim, por cada pacote de som de 32000 bytes (4 segundos), o servidor envia 64 datagramas de 500 bytes cada. Como cada datagrama tem 8 bytes de cabeçalho (*overhead*), a eficiência na transferência de um pacote de som não comprimido é de 98,4%. Este valor foi calculado através da fórmula (1):

$$eficiência = \frac{bytes\_uteis}{total\_de\_bytes\_transferidos} \quad (1)$$

No caso do TCP, o *overhead* existente, é o relativo à troca de mensagens de início e fim de sessão e o provocado pelos cabeçalhos dos segmentos TCP (20 bytes). No entanto, não é possível dar um valor de eficiência concreto para este método pois está dependente da segmentação dos

pacotes efectuada pelo sistema operativo e pelos comutadores (desprezando os *bytes* transferidos durante o inicio e fim de sessão). Deste modo, a eficiência deste protocolo para a transferência de um pacote de som de 32000 bytes, poderá ser de 99,88% se o pacote for enviado por inteiro, de 99,4% se for dividido em 8 segmentos ou inferior no caso de ser repartido em segmentos menores. Estes valores foram obtidos através da fórmula (1) adaptada para o método TCP, onde por cada pacote de som, transfere-se também um pacote de 4 bytes contendo a dimensão em bytes do áudio transferido.

Voltando a analisar a **Figura V.6**, o débito da rede UMTS durante a execução deste teste, parece variar consideravelmente entre cada pacote recebido. Isto pode acontecer porque de facto a rede apresenta oscilações, ou então porque, como estamos a lidar com valores reduzidos de tempo de recepção do pacote, a API utilizada, juntamente com o processamento efectuado pelo telemóvel das várias *threads* existentes na aplicação, não permitem a obtenção de uma quantidade exacta para este valor, pois qualquer variação deste, influência bastante o valor obtido para o débito da rede. No entanto, o valor médio obtido para o débito neste caso é de 100 kbps, o que até se aproxima dos 120 kbps calculados sobre a mesma rede (UMTS) a partir do método em TCP.

A compressão dos pacotes revela-se praticamente inexistente no caso de amostras de 500 *bytes*, o que faz com que a técnica utilizada não surta o efeito desejado. No entanto, uma situação não representada nestes gráficos é a da transferência de silêncio, onde se consegue comprimir este tipo de pacotes para valores abaixo dos 100 *bytes* (permitindo definir um limiar de silêncio que possa ser utilizado).

A descompressão dos pacotes de áudio mantém-se nos 15 ms para pacotes que vêm de facto comprimidos. Os que são transferidos com o tamanho original não passam por este processo. No lado do servidor a compressão dos pacotes de som é praticamente instantâneo.

Como os datagramas transferidos são armazenados em memória até que sejam agrupados em número suficiente para criar pacotes de 4 segundos para serem reproduzidos, o tempo de preparação dos *players* é, como no caso do TCP, na ordem dos 300 a 400 ms.

Apesar do *buffering* utilizado, implicar que um *player* só comece a reproduzir áudio quando já estiver disponível um outro pacote em memória para o próximo *player* começar a prepará-lo, a pequena pausa existente entre as alternâncias de *players* mantêm-se, não se conseguindo também neste caso obter um verdadeiro fluxo de som sem interrupções.

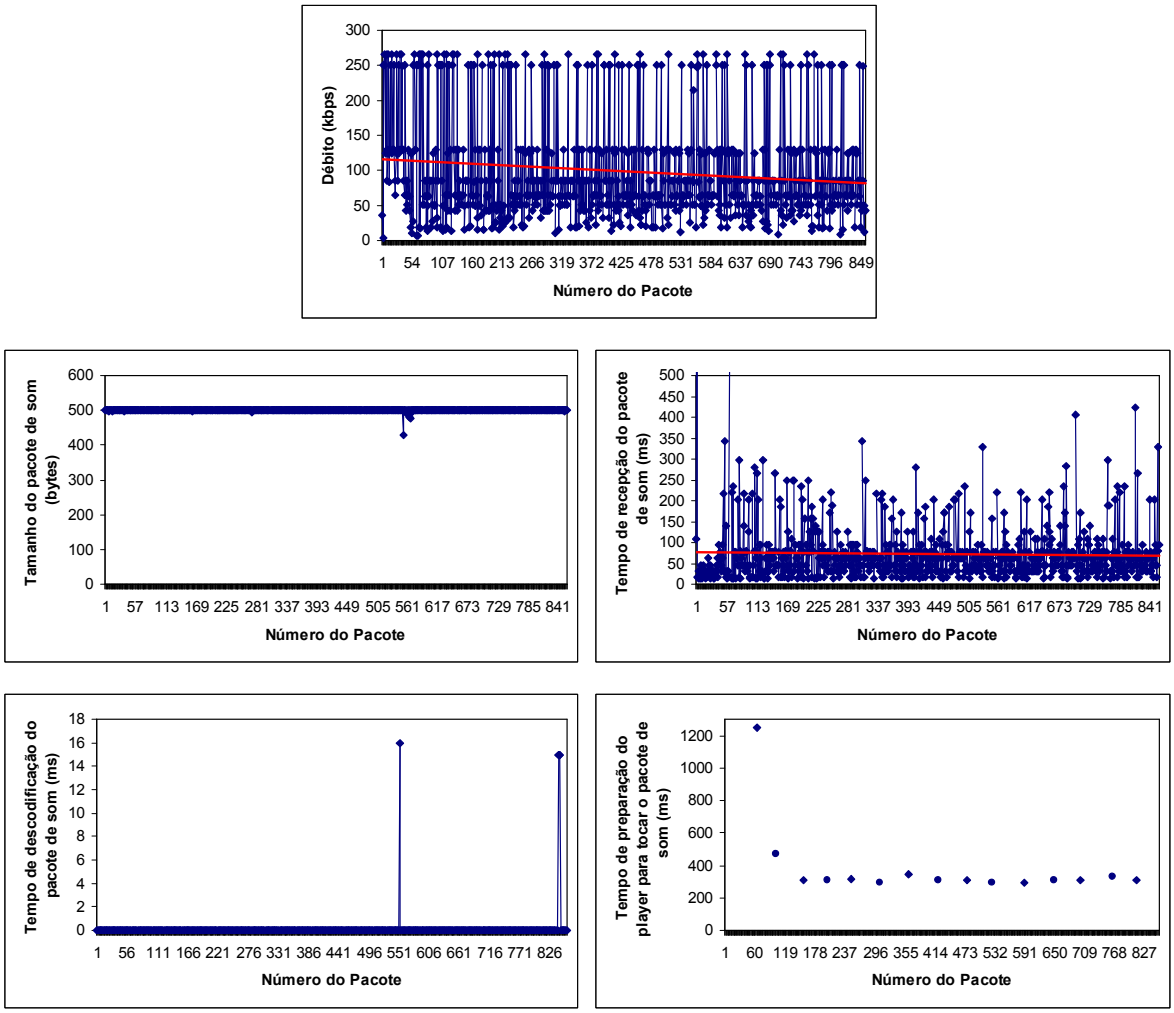


Figura V.6 - Gráficos relativos à transferência de som por UDP durante cerca de 1 minuto

Pela análise destes resultados, pelos factos apresentados na **Secção IV.3** e por se ter revelado uma solução mais robusta, resolveu-se escolher o método de transferência de som sobre TCP para a versão final da aplicação do cliente. Outro factor a favor desta opção, é que nem todos os modelos de telemóvel suportam datagramas UDP, estando o TCP mais generalizado.

V.5.2 Actualizações integrais ao ambiente de trabalho

Para se verificar a duração de uma actualização ao ambiente de trabalho nos vários modos disponíveis na aplicação, foram realizados testes sobre condições semelhantes para todos os modos. Assim, tentando analisar o funcionamento da aplicação numa situação extrema, cobriu-se todo o ambiente de trabalho com uma imagem de dimensões 1024 por 768, onde se podia verificar um elevado gradiente de cores utilizadas (**Figura V.7**), tendo sido efectuados 10 pedidos de actualização integrais por cada teste efectuado. Resolveu-se ainda utilizar a rede GPRS ao invés da UMTS para se verificar o débito disponível nesta e também para analisar o pior caso em termos de tempos de transmissão.

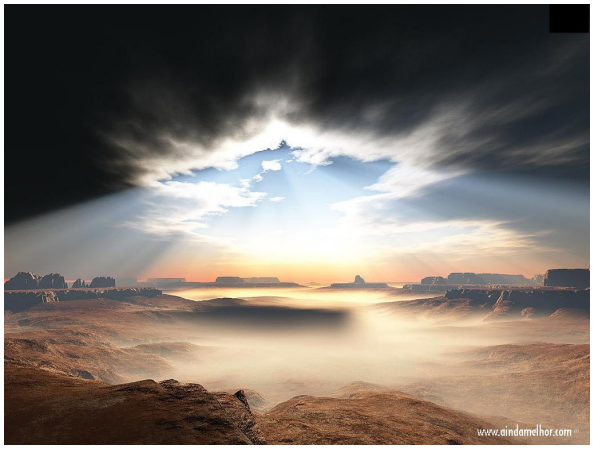


Figura V.7 – Imagem utilizada para os testes às actualizações ao ambiente de trabalho

Ao todo foram realizados 50 pedidos de actualização, podendo-se ver na **Figura V.8** que o valor médio registado para o débito foi de 56,4 kbps (com algumas oscilações), como seria de esperar na rede GPRS utilizada. O desvio padrão obtido foi de 4,5 e o intervalo de confiança a 95% é [55,2; 57,7]

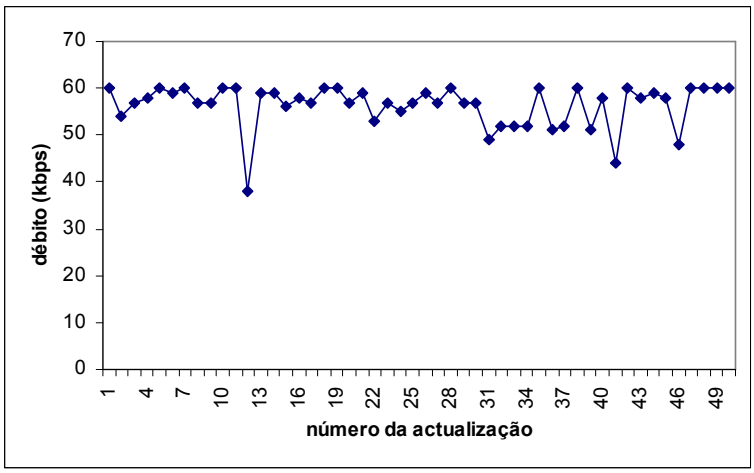


Figura V.8 – Débito obtido na rede GPRS

No modo em que as alterações ao factor de escala são efectuadas no telemóvel, é visível na **Figura V.9** que o facto de estar seleccionada ou não a opção de *smooth navigation* em pouco ou nada influência o tempo de recepção da actualização. Isto seria de esperar visto que em ambos os casos o tempo obtido apenas é influenciado pelo tempo de recepção e pelo tempo que o telemóvel leva a desenhar no ecrã e nos *buffer's* de memória respectivos, os três zoom's disponíveis. Poder-se-ia pensar que o tempo necessário para desenhar no *buffer* de memória correspondente ao zoom de 100% (*zoom normal*) seria menor no caso de não estar seleccionada a opção de *smooth navigation*, pois este apenas concentra a secção correspondente às dimensões do ecrã do telemóvel e não de todo o ambiente de trabalho, no entanto, isto só é verdade para actualizações parciais efectuadas na navegação neste zoom. Para actualizações integrais, devido à implementação utilizada, despende-se o mesmo tempo a desenhar em ambos os modos.

Pode-se ainda verificar que para a imagem de fundo utilizada que ocupa 108278 *bytes* no servidor, foram recebidos no telemóvel 227309 *bytes* referentes à actualização, em cerca de 30 segundos. Todo este *overhead* deve-se às codificações utilizadas e ao formato das mensagens utilizado pelo protocolo RFB para enviar os dados referentes às actualizações.

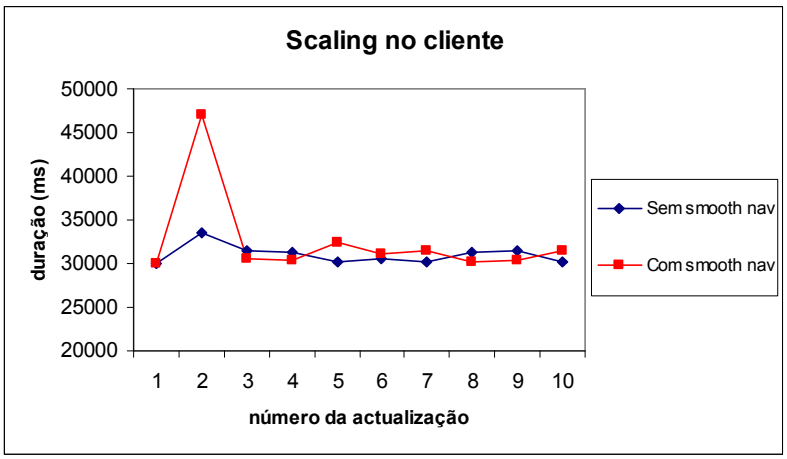


Figura V.9 – Duração das actualizações quando a alteração ao factor de escala é realizada no telemóvel.

No caso em que a alteração ao factor de escala é efectuada no lado do servidor (**Figura V.10**), apenas se testou o modo com a opção de *smooth navigation* seleccionada, pois só assim se conseguem obter actualizações integrais ao ambiente de trabalho, mantendo a coerência com os testes efectuados previamente. Neste caso, o único *buffer* de memória utilizado para desenhar no ecrã do telemóvel é o de *zoom normal* e por isso seria de esperar que a duração das actualizações fossem menores do que no caso anterior, por não se ter de reduzir no telemóvel o ambiente de trabalho para os outros factores de escala. No entanto, isto não se verifica e o tempo dispendido na actualização é aproximadamente o mesmo que no caso anterior (30 segundos). Conclui-se assim, que o processamento efectuado por parte do cliente pode ser negligenciado, pelo menos no telemóvel utilizado para testes (Nokia 6630 com processador RISC de 32-bits a 220 MHz). Um ponto a favor da alteração ao factor de escala ser efectuada no lado do servidor, é que as actualizações

para *zoom's* inferiores a 100% tornam as actualizações mais rápidas. Reduzindo para metade o factor de escala, obtém-se uma redução de 3 vezes para o tempo gasto na actualização. Para uma redução de 3 vezes do factor de escala obtém-se uma redução de 6 vezes no mesmo tempo. Apesar disto implicar actualizações, para *zoom's* menores que 100%, mais rápidas do que as obtidas quando as alterações ao factor de escala são efectuados no telemóvel, neste último caso, uma mudança de factor de escala não implica um pedido de actualização e por consequente poupa-se algum tempo neste aspecto. Deste modo, não se pode eleger um modo como sendo melhor que o outro, cabendo ao utilizador esta escolha.

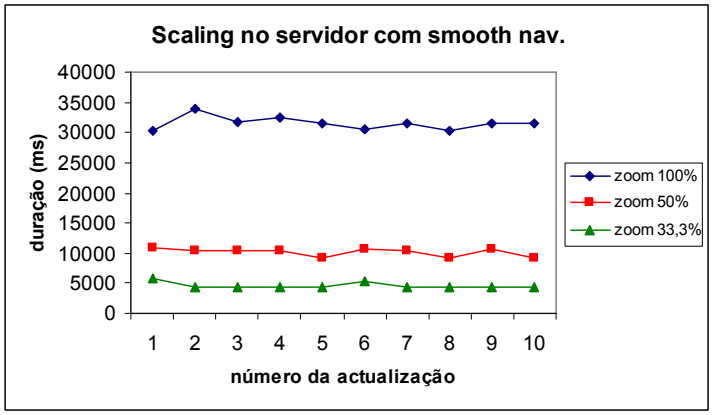


Figura V.10 – Duração das actualizações quando a alteração ao factor de escala é realizada no lado do servidor

Estes valores são no entanto apenas indicativos para a situação em questão, pois para ambientes de trabalho menos complexos, os tempos gastos por actualização em GPRS, podem baixar para valores entre 5 a 15 segundos e até menos para uma ligação sobre UMTS. O facto de se realizarem actualizações integrais também é um caso extremo, porque, tirando a actualização inicial e algum pedido para tal por parte do utilizador, a maioria das actualizações serão incrementais, ou seja apenas serão transferidas as porções do ambiente de trabalho que sofreram alterações. Por consequente, os tempos associados a este tipo de actualizações serão consideravelmente menores.

Utilizando um telemóvel/PDA HTC TyTn (processador *Samsung* a 400Mhz) para realizar o mesmo ensaio, deparou-se com um tempo dispendido por actualização bastante superior. Neste caso, obtiveram-se tempos na ordem dos 7 minutos ao invés dos 30 segundos obtidos no Nokia 6630. Para analisar se o motivo para tal acontecimento num dispositivo de nível superior se deve ao desenhar no ecrã, da imagem visível, foram efectuados testes em que o ambiente de trabalho vai ou não sendo desenhado no ecrã à medida que a actualização é recebida. Obteve-se no primeiro caso tempos a rondar os 7 minutos e 30 segundos e no segundo caso 6 minutos e 55 segundos, o que revela um incremento de velocidade na ordem dos 8%. Dado os valores em questão (e o desespero do utilizador), confirmou-se assim que a diferença entre a imagem ir sendo ou não redesenhada no ecrã ao longo da actualização, não é um factor crucial e acarreta o benefício do utilizador estar a par do estado da actualização. Deste modo, a única razão encontrada para justificar tal ocorrência prende-se com o facto de a máquina virtual Java ser parte integrante do sistema operativo do Nokia

6630 (*Symbian*) e não o ser no caso do HTC TyTn, existindo uma aplicação a correr sobre o sistema operativo Windows Mobile 6® que faz a gestão dos MIDlets Java, o que poderá tornar mais lento o processamento dos dados, tal como acontece nos emuladores utilizados para testes nos computadores.

V.5.3 Exemplo de um possível cenário de aplicação

Com o intuito de simular o funcionamento da aplicação desenvolvida num possível cenário real de utilização da mesma, foi concebida uma situação simples em que o utilizador possa sentir a necessidade de recorrer ao controlo remoto do seu computador pessoal. Deste modo, imagine-se que o utilizador se encontra em deslocação como passageiro de um veículo (sem acesso a computadores) e lembra-se que precisa de enviar um documento urgente à pessoa X mas que esse documento se encontra no seu computador de casa. Os passos necessários que o utilizador tem de efectuar para realizar tal tarefa passam por:

1. Aceder ao computador remoto
2. Abrir o documento em questão (neste caso um ficheiro Word de uma página)
3. Dar uma revisão geral rápida e acrescentar uma pequena frase ao mesmo
4. Gravar o documento e fechá-lo
5. Abrir o *Thunderbird*
6. Criar novo e-mail, colocar assunto, destinatário e anexar o documento
7. Enviar

Este teste foi realizado utilizando a rede GPRS e o telemóvel Nokia 6630, sendo os resultados obtidos apresentados na **Figura V.11**. Todos os pedidos de actualizações foram incrementais excepto o inicial. Através de uma primeira observação dos gráficos resultantes, constata-se que os valores obtidos para o débito da rede oscilam bastante. Tal ocorrência deve-se ao facto da maioria dos pacotes transferidos serem de dimensões reduzidas, do tempo de recepção destes ser afectado pelo processamento e correspondente desenhar no ecrã do telemóvel, e pela baixa precisão do relógio interno do Java. É também evidente, que a dimensão das actualizações efectuadas varia ao longo do teste assim como o tempo dispendido na recepção e processamento das mesmas. Pode-se ainda perceber, através da análise destes gráficos, o que o utilizador está a realizar nos instantes da recepção dos pacotes. Por exemplo, o primeiro pacote (63898 *bytes*) representa a actualização inicial ao ambiente de trabalho remoto. O quarto pacote (33069 *bytes*) indica que o documento Word foi aberto e por consequente uma maior área do ambiente de trabalho remoto sofreu alterações. Seguem-se actualizações de dimensões reduzidas (entre 230 e 5419 *bytes*) que indicam alterações à posição do cursor, selecção de texto, escrita de uma pequena frase e mudança de linha, etc. Depois de gravar o documento, deparamo-nos com algumas actualizações de dimensões superiores (entre 22006 e 57395 *bytes*) que representam o fecho do documento de Word, a alteração para a aplicação *Thunderbird* e o abrir da janela referente à criação do novo e-mail. Sucedem-se várias actualizações referentes a deslocamentos do cursor e à escrita do destinatário e do assunto (entre 159 e 3500 *bytes*) e uma actualização referente ao abrir da janela onde se escolhe o documento a anexar ao e-

mail (23428 *bytes*). Facilmente se interpretarão também as restantes actualizações efectuadas mas tentou-se apenas dar aqui uma ideia da análise dos gráficos. Verifica-se ainda que o tempo de recepção dos pacotes tende a ser directamente proporcional às dimensões das actualizações, mas nota-se que em alguns casos, o tempo de recepção é maior que o suposto, possivelmente devido a atrasos introduzidos pelo processamento dos pacotes por parte do telemóvel. Toda a simulação deste cenário demorou cerca de 9 minutos a ser realizada, revelando que manejar a interface exígua do telemóvel para controlar remotamente um computador exige alguma paciência por parte do utilizador.

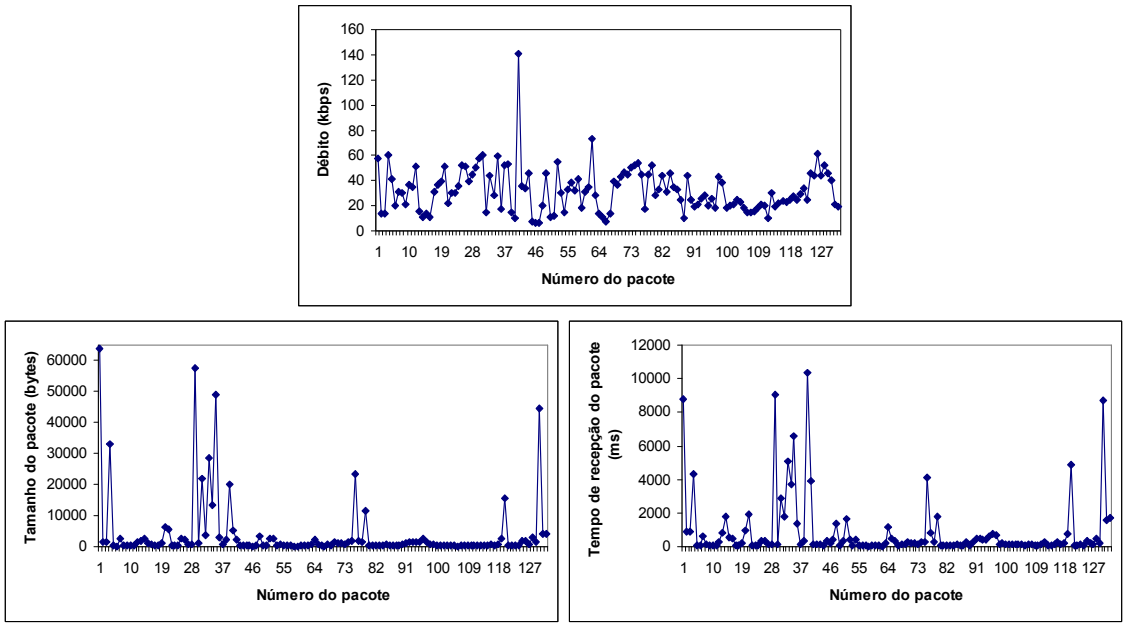


Figura V.11 – Gráficos relativos à utilização da aplicação num cenário típico.

V.5.4 Tempos de Ida e Volta

Não sendo um factor crucial neste tipo de aplicação, é no entanto importante que uma acção realizada sobre o computador remoto se repercuta neste o mais rapidamente possível. Deste modo, para se ter uma ideia dos valores envolvidos em termos dos tempos de ida e volta existentes nas redes móveis celulares, foram realizados 5 *Pings* ao servidor utilizando ambos os métodos implementados (sobre TCP e UDP). Obtiveram-se 25 valores para o tempo de ida e volta no caso da implementação em TCP e mais 25 no caso da implementação em UDP. Posteriormente, foi realizada uma média destas estimativas e foram retirados os seus valores mínimos e máximos e apresentados na **Tabela V.1** e na **Tabela V.2**.

Tabela V.1 – Tempos de Ida e Volta obtidos no dispositivo HTC TyTn.

HTC TyTn	TCP (min /médio / max)	UDP (min / médio / máx)
GPRS (ms)	4086 / 4414 / 5887	617 / 684 / 869
UMTS (ms)	2832 / 3099 / 4040	354 / 428 / 473
HSDPA (ms)	1940 / 2205 / 3223	145 / 171 / 242

Tabela V.2 – Tempos de Ida e Volta obtidos no dispositivo Nokia 6630.

Nokia 6630	TCP (min /médio / máx)	UDP (min / médio / máx)
UMTS (ms)	437 / 672 / 1391	312 / 502 / 828

Os valores teóricos esperados para a rede GPRS são de 600 a 700 ms alcançando por vezes um segundo [19]. Para a rede HSDPA os valores esperados são de 100 a 200 ms e a rede UMTS sofre um incremento de aproximadamente 100 ms sobre a HSDPA [20]. Comparando os valores teóricos com os valores obtidos sobre a implementação em UDP, verifica-se que os resultados auferidos encontram-se dentro da gama prevista, com excepção das estimativas sobre UMTS que se revelam superiores ao esperado em cerca de 200 ms. Este facto pode dever-se à distância do terminal à estação de base, à implementação do protocolo UMTS utilizada no dispositivo móvel e às capacidades deste, etc. Em termos das estimativas obtidas com a implementação do *Ping* em TCP estas revelaram-se bastante dispares dos valores teóricos. Outro facto interessante é que ao contrário do que acontece com a implementação em UDP que mantém a coerência das estimativas obtidas entre dispositivos, em TCP o dispositivo HTC TyTn apresentou valores bastante superiores aos obtidos no telemóvel Nokia 6630 (na ordem dos poucos segundos). Tal ocorrência pode dever-se às diferentes implementações do protocolo TCP/IP nos sistemas operativos distintos que suportam os dispositivos em questão. Tal ordem de grandeza explica os atrasos existentes entre os pedidos efectuados pelo terminal móvel e as correspondentes respostas do servidor mas não revela nenhuma razão para o tempo de recepção das actualizações no HTC TyTn serem tão demoradas, pois após o atraso inicial introduzido pela rede, toda a informação transferida é recebida prontamente, como é sugerido pela boa performance obtida na transferência de som.

V.5.5 Memória requerida para o funcionamento da aplicação

Sendo este um aspecto importante a ter em consideração dadas as limitações dos dispositivos móveis, não foi no entanto possível obter valores representativos dos requerimentos da aplicação. Uma das razões para tal deve-se ao facto da memória requerida pela aplicação depender das dimensões díspares dos ecrãs dos telemóveis, do número de cores disponíveis, dos diferentes mecanismos de gestão de memória utilizados pelos aparelhos, etc. Nomeadamente, não foi possível obter valores concretos para a memória utilizada pelo telemóvel Nokia 6630 nos diversos modos disponíveis, pois este utiliza uma gestão de memória dinâmica, aumentando ou diminuindo a memória disponível para a aplicação consoante as necessidades desta. Verificou-se no entanto que este dispositivo, utilizando a memória inerente ao mesmo (sem a utilização de cartões de memória) não suporta aceder a servidores com resoluções acima dos 1024x768 com o modo de *smooth*

navigation seleccionado. O mesmo já não se verifica com o dispositivo HTC TyTn onde foram efectuadas ligações a servidores com resoluções até 2048x1536 utilizando todos os modos disponíveis. No caso deste terminal móvel, a gestão de memória já não é dinâmica e foi possível obter valores para a memória utilizada pela aplicação nos diversos modos e sobre várias resoluções. Verificou-se, como era esperado, que a memória utilizada pelos *buffers* referentes ao *zoom fifty* (50%) e ao *zoom normal* (100%) com *smooth navigation*, está directamente relacionada com a resolução utilizada e que a memória ocupada pelo segundo é cerca de 4 vezes superior ao primeiro. Os *buffers* de *zoom out* e de *zoom normal* sem *smooth navigation* utilizam sempre a mesma quantidade de memória pois não dependem da resolução do servidor. Apesar da informação obtida através dos testes efectuados, resolveu-se não apresentar aqui valores específicos, pois estes não se revelaram representativos. Para além disso, os valores adquiridos foram bastante díspares, em termos de ordem de grandeza, dos obtidos no Nokia 6630.

Capítulo VI. Conclusões

VI.1 Conclusões

O presente trabalho abordou de forma abrangente, o problema de proporcionar ao utilizador de um terminal móvel o controlo remoto sobre um computador. Iniciou-se a abordagem ao tema pela análise das soluções existentes no mercado, realizando o estudo das suas características e funcionalidades, cogitando possíveis contribuições para as mesmas e reflectindo sobre métodos alternativos para obter resultados semelhantes. Posteriormente, a intervenção efectuada abordou as seguintes contribuições:

1. Melhoramentos à aplicação existente, tornando a sua utilização mais simples, eficiente e apelativa ao utilizador. Incorporação de novas funcionalidades de modo a tornar a aplicação escolhida uma alternativa viável às soluções comerciais.
2. Implementação de métodos alternativos para alteração ao factor de escala da imagem visível no ecrã do telemóvel e de navegação pelo ambiente de trabalho remoto. Foram disponibilizados vários modos de funcionamento para dar suporte a dispositivos com diferentes capacidades de memória e processamento.
3. Concepção de um método para transferência de som do servidor para o dispositivo móvel, com compressão de dados e supressão de silêncio, tendo em consideração as limitações impostas pela maioria dos terminais.
4. Criação de um mecanismo de perfis de utilizador que permite configurar no servidor determinadas características, diferenciar clientes e dar acesso aos mesmos a aplicações específicas.
5. Realização de testes para estudo da viabilidade do projecto sobre as várias redes móveis celulares existentes e para estudo das características das mesmas.

Durante todo o projecto, um factor valorizado que se teve em consideração foi a de manter a aplicação resultante compatível com o maior número de plataformas móveis possíveis. Manteve-se também a aplicação compatível com as diversas versões de servidores VNC existentes, com o prejudico de não estarem disponível algumas das funcionalidades desenvolvidas sobre a versão do servidor adoptada.

Dos testes efectuados, revelou-se que as redes móveis celulares existentes possuem características suficientes para o correcto funcionamento da aplicação. No entanto, para a transferência de som revelou-se necessária uma largura de banda superior a 64 kbps que só é obtida pelas redes UMTS e HSDPA. Em termos dos clientes, espera-se que dentro de poucos anos as interfaces inerentes aos terminais móveis estejam mais maduras para aplicações como esta, com ecrãs de maiores dimensões, melhores resoluções e mecanismos de entrada mais simples para o utilizador.

VI.2 Trabalho Futuro

Em cada área deste trabalho, existem novas funcionalidades que podem ser integradas para produzir um sistema cada vez mais robusto e apelativo ao utilizador. Só a título de exemplo são de seguida apresentados alguns tópicos que ficaram por abordar:

- Poder-se-ia permitir ao utilizador associar as teclas do telemóvel às do PC ou a combinações destas, sem o utilizador ter de ficar limitado às teclas predefinidas no código da aplicação. Por exemplo, um utilizador que navegue exaustivamente por documentos de texto, se calhar gostava de ter as teclas de PgUp/PgDn associadas a teclas específicas do telemóvel, para evitar ter de ir constantemente ao formulário *Special Keys*. Este aspecto também se revelaria interessante para dispositivos com teclados qwerty que normalmente dispõem de teclas extras, que o utilizador poderia associar às funções que preferisse.

- Permitir enviar e/ou receber ficheiros para, por exemplo, permitir ao utilizador descarregar um documento importante que precisa na altura e que deixou no computador de casa, ou simplesmente para transferir uma música que lhe apetece ouvir no momento, como acontece na aplicação proprietária RDM+. Actualmente, as únicas coisas que são armazenadas no telemóvel são o estado das opções disponíveis ao utilizador no formulário inicial, os endereços dos sistemas anfitriões e correspondentes palavras-chave e as macros definidas no Midlet *About*. Para isto é utilizado o RMS (*Record Management System*) disponibilizado pelo J2ME, que permite de formar simples, armazenar dados no telemóvel num ficheiro de formato especial, sem o programador se ter de preocupar com o acesso à memória ou com assuntos relacionados com a segurança. O mesmo não acontece quando se pretende armazenar um ficheiro completamente independente algures na memória flash do telemóvel porque o simples facto de se querer aceder, criar e/ou apagar ficheiros no terminal móvel traz problemas de segurança, especialmente quando se lida com fabricantes diferentes.
  Muitos fabricantes produziram API's proprietárias para aceder aos ficheiros locais no telemóvel, no entanto, em quase todos os casos de acesso a ficheiros, a aplicação precisa de estar assinada (*signed*), para garantir que esta possa aceder aos mesmos. Como qualquer aplicação Java, um Midlet corre num "caixa de areia" segura e não pode simplesmente aceder ao sistema de ficheiros ou a outra qualquer informação "privada". Se o dispositivo suporta a API opcional *FileConnection*, pode-se aceder ao sistema de ficheiros, no entanto, esta está disponível, por exemplo, no Nokia 6630 mas não no Nokia 6600. Por isso, e como queremos uma aplicação compatível com o maior número de plataformas móveis possíveis, esta não seria a solução adequada e outra teria de ser equacionada.

- Expandir as codificações utilizadas pelo protocolo RFB e realizar um estudo das que melhor se adequam às plataformas móveis em termos de processamento necessário e de tráfego gerado na rede.

- Ao nível da implementação do som seria bom construí-la segundo os mesmos princípios do VNC, ou seja, inicialmente assumir-se o mínimo sobre a rede de dados que suporta a comunicação e sobre os *players* disponíveis no cliente (tal como foi implementado) mas permitindo negociar-se uma melhor qualidade de áudio transferido e mesmo diferentes formatos de codificação. Dado que algumas plataformas móveis mais recentes já começam a suportar *streaming* em RTSP sobre o J2ME, poder-se-ia implementar também esta solução e permitir ao utilizador a selecção do método de transferência de som que mais se adequasse ao seu dispositivo.

  Uma outra característica que podia ser interessante oferecer era a possibilidade da transferência de som ser bidireccional. Isto permitia, por exemplo, no caso de se estar a efectuar uma assistência remota, que o auxiliado e o auxiliando pudessem comunicar entre si.

- Exportar para outras plataformas (Linux, Mac…) as novas funcionalidades introduzidas no servidor de VNC para Windows.

- Cifrar toda a informação trocada entre cliente e servidor. Neste momento, toda a informação trocada depois de estabelecida uma sessão VNC pode ser capturada por alguém alheio à comunicação e deste modo ter acesso a informação "privada". O único mecanismo de segurança existente neste momento encontra-se no início de sessão para autenticar o utilizador através da palavra-chave, sendo esta informação trocada de forma segura. Para proteger de um eventual furto do telemóvel, o acesso ao livro de endereços, onde são armazenados os sistemas anfitriões, está também protegido através de uma palavra-chave definida pelo utilizador.

- Mantendo todas as características que fazem desta aplicação compatível com um maior número de plataformas móveis possíveis, seria interessante introduzir a hipótese de utilização de mecanismos específicos para determinados telemóveis (por exemplo a utilização de streaming), que melhorassem a performance da aplicação.

Referências

[1] Tristan Richardson, "*Virtual Network Computing*", *IEEE Internet Computing*, Jan/Feb 1998.

[2] Tristan Richardson, "*The RFB Protocol*" version 3.8, PDF, June 2007.

[3] "*How to realise the benefits of mobile broadband today*", Global System for Mobile Communications (GSMA) publication, February 2007.

[4] Comparação das soluções da ShapeServices. Último acesso em 10/08/2007.
http://www.shapeservices.com/en/products/remote_whitepaper.php

[5] Microsoft WAVE sound file format. Último acesso em10/08/2007.
http://ccrma.stanford.edu/CCRMA/Courses/422/projects/WaveFormat/

[6] Microsoft corporation, "*Windows Server 2003 Remote Access Overview*", White Paper, 1-2, March 2003.

[7] Microsoft, *"INFO: UDP Datagram Can Be Silently Discarded if Larger than MTU"*
http://support.microsoft.com/kb/233401 Último acesso em 10/08/2007

[8] Oliveira, J., Kamienski, C. A., Kelner, J., Sadok, D., "Análise de Desempenho de TCP sobre GPRS em um Ambiente Fim a Fim", Workshop de Comunicação Sem Fios (WCSF 2004), Fortaleza/CE, October 2004.

[9] R. Chakravorty, J. Cartwright and I. Pratt, "*Practical Experience with TCP over GPRS*", *Proc. of IEEE GLOBECOM 2002*, November 2002

[10] "*TCP over second (2,5G) and third (3G) Generation Mobile Networks*", RFC 3481, February 2003

[11] Antonis Alexiou, Christos Bouras, Vaggelis Igglesis, "*Performance Evaluation of TCP over UMTS Transport Channels*", Whitepaper

[12] Sun J2ME, "*Mobile Media API (MMAPI); JSR 135 Specification*", June 2006
http://java.sun.com/products/mmapi/ Último acesso em 28/08/2007.

[13] Sun J2ME, "*Mobile Information Device Profile (MIDP1.0); JSR 37 Specification*", December 2000.
http://java.sun.com/products/midp/ Último acesso em 28/08/2007.

[14] Eduardo Tude, "Tutorial de GPRS", (www.teleco.com.br), April 2003

[15] Eduardo Tude, "Tutorial de UMTS", (www.teleco.com.br), January 2004

[16] Eduardo Tude, "Tutorial de HSDPA", (www.teleco.com.br), February 2005

[17] Michael Lloyd Lee, "J2ME VNC", código fonte, February 2005.
http://j2mevnc.cvs.sourceforge.net/j2mevnc/VNC/ Último acesso em 10/08/2007.

[18] AT&T, Harakan Software, "*WinVNC v3.3.3 r7 with Server Scaling Extensions*", source code, January 2001. http://www.btinternet.com/~harakan/PalmVNC/ Último acesso em 10/08/2007.

[19] "*General Packet Radio Service (GPRS)*", Wikipédia article, August 2007
http://en.wikipedia.org/wiki/General_Packet_Radio_Service Último acesso em 28/08/2007.

[20] V. Rivoira and F. Pascali, "HSDPA: High-speed internet over your mobile phone", IEC newsletter, June 2007. http://www.answers.com/topic/gprs?cat=technology Último acesso em 28/08/2007.

[21] John W. Muchow, "CORE J2ME - TECNOLOGIA E MIDP", Makron Books, 2004

[22] Microsoft, "Microsoft Developer Network (MSDN)", http://msdn2.microsoft.com/en-us/default.aspx

[23] 3rd Generation Partnership Project (3GPP), http://www.3gpp.org/

[24] GNU General Public License, http://www.gnu.org/licenses/gpl.html

[25] Global System for Mobile communications (GSM), http://www.gsmworld.com/

[26] Institute of Electrical and Electronics Engineers (IEEE), http://www.ieee.org/

[27] Sun, Java Platform, Micro Edition (Java ME), http://java.sun.com/javame/index.jsp

[28] Eric Giguere, "*Databases and MIDP, Part 1: Understanding the Record Management System*", Sun article, February 2004. http://developers.sun.com/mobility/midp/articles/databaserms/index.html

[29] International Telecommunication Union (ITU/UIT), http://www.itu.int/

[30] Hélio Candeias, "Sistema de Tele-Vigilância Suportado em GSM, GPRS, CDMA2000 e UMTS", Trabalho final de curso (ISEL), 6-16, April 2006.

Anexo A. Protocolo RFB

A.1 Protocolo de *Display*

A parte de *display* deste protocolo baseia-se numa única primitiva gráfica: *Colocar um rectângulo de pixeis na posição (x, y) especificada*. Numa primeira aproximação isto pode parecer uma forma ineficiente de desenhar as coisas, no entanto, disponibilizando vários tipos de codificação para os *pixeis*, isto permite bastante flexibilidade em parâmetros como sendo a largura de banda, rapidez de desenhar do cliente e velocidade de processamento do servidor.

Uma sequência destes rectângulos faz uma actualização (*update*). Uma actualização representa uma mudança de um estado do *framebuffer* válido, para outro, portanto, em certos aspectos é parecido com uma *frame* de vídeo.

As actualizações são requisitadas pelo cliente, ou seja, uma actualização só é enviada do servidor para o cliente, em resposta a um pedido explícito do cliente. Isto dá ao protocolo uma qualidade adaptativa, pois quanto mais lento o cliente e a rede forem, mais lento será o ritmo de actualizações. Por exemplo, ao arrastar uma janela no ambiente de trabalho as alterações no *framebuffer* acontecem umas a seguir às outras, mas com um cliente ou rede lenta, os estados transientes podem ser ignorados, o que resulta em menos tráfego na rede e em menos processamento e correspondente desenhar no ecrã por parte do cliente (este pode só desenhar a janela na sua posição inicial e depois na sua posição final).

A.2 Protocolo de entrada (*input*)

A parte de entrada deste protocolo baseia-se nos modelos standard de um teclado e de um rato multi-botões (ou dispositivos análogos). O cliente envia eventos ao servidor consoante o utilizador carrega nas teclas ou mexe no rato e o servidor faz o processamento desta informação e actua correspondentemente. Estas entradas podem também ser sintetizadas de outros dispositivos de entrada/saída não standard, como por exemplo, utilizando uma caneta (*pen*) e um software de reconhecimento de caligrafia para gerar eventos de teclado.

A.3 Representação de um *pixel*

A interacção inicial entre o cliente e o servidor envolve a negociação do formato e codificação utilizada para representar os *pixeis* transmitidos. Esta negociação foi concebida para tornar o trabalho do cliente o mais simples possível, ou seja, o servidor deve poder sempre fornecer *pixeis* na forma que o cliente quer.

O formato dos *pixeis* refere-se à representação das cores individuais pelos valores destes. Os formatos mais comuns são os de 24-bits ou de 16-bits (*true colour*), onde os valores dos *pixeis* são traduzidos directamente para as intensidades de vermelho, verde e azul, e os de 8-bits (mapa de cores) onde um mapeamento arbitrário pode ser utilizado para traduzir o valor dos *pixeis* para as intensidades das cores.

A codificação refere-se a como é que um rectângulo de *pixeis* é enviado pela rede. Cada rectângulo de *pixeis* é precedido por um cabeçalho que indica a posição x, y do rectângulo no ecrã, o seu comprimento, a sua altura e o tipo de codificação utilizada. O conteúdo destes segue depois a codificação especificada.

Os tipos de codificação definidos actualmente são: *Raw*, *CopyRect*, *RRE*, *CoRRE, Hextile* e *ZRLE*. Entre estes os mais usados são o *ZRLE*, *Hextile*, e *CopyRect* pois providenciam a melhor compressão para os ambientes de trabalho típicos.

A.4 Extensões ao protocolo

Existem várias formas por onde se podem fazer extensões ao protocolo:

Novas codificações, podem ser facilmente adicionadas ao protocolo, mantendo a compatibilidade com os clientes e servidores existentes, pois os servidores irão simplesmente ignorar pedidos de codificações que não suportem, e os clientes nunca pedirão essas novas codificações.

Pseudo codificações, podem ser pedidas pelos clientes para declarar ao servidor que suportam uma certa extensão ao protocolo. Um servidor que não suporte a extensão irá simplesmente ignorá-la, o que significa que o cliente deve assumir que o servidor não suporta a extensão em causa até receber uma confirmação específica deste.

Novos tipos de segurança podem ser adicionados, dando assim uma grande flexibilidade ao comportamento do protocolo sem sacrificar a compatibilidade existente. Um cliente e servidor que acordem um novo tipo de segurança, podem efectivamente comunicar usando qualquer protocolo depois disso (não tendo de ser necessariamente o protocolo RFB).

Apenas as versões de protocolo são rígidas e não podem ser usados números de versões diferentes dos que estão definidos pela *RealVNC Ltd*. Se o programador utilizar um número de versão diferente, então nem se é compatível *com o* RFB nem com o VNC.

A.5 Mensagens de Protocolo

Existem duas fases do protocolo: uma fase inicial de *handshaking*, seguido da interacção normal do protocolo.

O *handshaking* inicial consiste nas mensagens de *Versão do Protocolo* (enviado pelo cliente e pelo servidor), *Segurança*, *Inicialização do Cliente* e de *Inicialização do Servidor*.

O protocolo depois prossegue com a fase de interacção normal onde o cliente pode enviar as mensagens que pretender e pode receber mensagens do servidor como resultado destas. Todas estas mensagens começam com um byte *message-type*, seguido dos dados específicos do tipo de mensagem.

Os tipos de mensagens que serão abordados nos pontos seguintes usam os tipos básicos U8, U16, U32, S8, S16, S32, que representam respectivamente 8, 16 e 32-bit *unsigned integer* e 8, 16 e 32-bit *signed integer*.

O tipo PIXEL significa um valor de *pixel* de *bytesPerPixel* bytes, onde 8 x *bytesPerPixel* é o número de bits por *pixel* acordados pelo cliente e servidor (na mensagem de *Inicialização do Servidor* ou na mensagem de *Definir o formato dos Pixeis*).

A.5.1 Mensagens de *handshaking* inicial

A.5.1.1 Versão do Protocolo

O *handshaking* começa com o servidor a enviar ao cliente a mensagem de *Versão do Protocolo*. Esta faz com que o cliente saiba qual é o número de versão do protocolo RFB mais alta que o servidor suporta. O cliente depois responde com uma mensagem semelhante indicando o número da versão que deve ser usada, podendo ser igual ou inferior à enviada pelo servidor.

As versões do protocolo existentes actualmente são as 3.3, 3.7, 3.8, sendo que a adição de uma nova codificação ou pseudo-codificação não requer uma mudança da versão do protocolo, já que o servidor pode sempre ignorar as codificações que não suporta.

A mensagem *Versão do Protocolo* consiste numa cadeia de caracteres ASCII de 12 bytes no formato "RFB xxx.yyy\n" onde o xxx representa a parte inteira da versão e o yyy representa a parte fraccionária desta, preenchidos (*padded*) com zeros.

A.5.1.2 Segurança

Assim que a versão do protocolo utilizada está decidida, o cliente e o servidor têm de concordar no tipo de segurança que vão utilizar na ligação.

Versão 3.7 e 3.8 O servidor lista os tipos de segurança que suporta:

Numero de bytes	Tipo [Valor]	Descrição
1	U8	*número-de-tipos-de-segurança*
número-de-tipos-de-segurança	Vector de U8	*tipos de segurança*

Se o servidor listou pelo menos um tipo de segurança suportado pelo cliente, então o cliente envia de volta um único byte indicando qual o tipo de segurança a ser usado na ligação:

Numero de bytes	Tipo [Valor]	Descrição
1	U8	*tipo de segurança*

Se o *tipo de segurança* é zero, então por alguma razão a ligação falhou. Isto é seguido por uma cadeia de caracteres descrevendo a razão para tal ter acontecido (a cadeia de caracteres é especificada como sendo o comprimento desta, seguido desse mesmo numero de caracteres ASCII).

Numero de bytes	Tipo [Valor]	Descrição
4	U32	*comprimento da razão*
comprimento da razão	Vector de U8	*cadeia de caracteres descrevendo a razão*

O servidor termina a ligação depois de enviar a *cadeia de caracteres descrevendo a razão*.

Versão 3.3 O servidor decide o tipo de segurança a ser usado e envia-o ao cliente:

Numero de bytes	Tipo [Valor]	Descrição
4	U32	*tipo de segurança*

O valor zero significa que a ligação falhou e é seguido por uma cadeia de caracteres descrevendo a razão conforme descrito anteriormente. Os *tipos de segurança* definidos são:

Numero	**Tipo**
0	Inválido
1	Nenhum
2	Autenticação do VNC
5	RA2
6	RA2ne
16	Tight
17	Ultra
18	TLS

Quando o *tipo de segurança* é decidido, os dados específicos para esse *tipo de segurança* são enviados de seguida. Depois – versões 3.8 e posteriores - o servidor envia uma mensagem ao cliente informando se foi bem sucedida a troca de mensagens de segurança.

Numero de bytes	Tipo [Valor]	Descrição
4	U32	*status:*
	0	***OK***
	1	***failed***

Note-se que depois da fase de troca de mensagens de segurança, é possível que os restantes dados do protocolo sejam trocados por um canal cifrado ou de qualquer outra forma alterado. No caso de ser bem sucedido, o protocolo continua com a mensagem de *Inicialização do cliente*.

A.5.1.3 Inicialização do cliente

Depois de o cliente e o servidor acordarem o tipo de segurança, o cliente envia uma mensagem de inicialização:

Numero de bytes	Tipo [Valor]	Descrição
1	U8	*flag partilha de desktop*

A *flag partilha de desktop* será diferente de zero se o servidor deve tentar partilhar o ambiente de trabalho, deixando outros clientes ligados e será zero se o servidor deve dar acesso exclusivo ao cliente, terminando a ligação com todos os outros clientes já ligados.

A.5.1.4 Inicialização do servidor

Depois de receber a mensagem de *Inicialização do cliente*, o servidor envia-lhe uma mensagem de inicialização indicando as dimensões do seu *framebuffer*, o formato dos *pixeis* que pretende utilizar e o nome associado ao ambiente de trabalho:

Numero de bytes	Tipo [Valor]	Descrição
2	U16	*largura do framebuffer*
2	U16	*altura do framebuffer*
16	PIXEL_FORMAT	*formato de pixel do servidor*
4	U32	*comprimento do nome*
comprimento do nome	Vector de U8	*cadeia de caracteres do nome*

Onde PIXEL_FORMAT é:

Numero de bytes	Tipo [Valor]	Descrição
1	U8	*bits por pixel*
1	U8	*profundidade*
1	U8	*flag big-endian*
1	U8	*flag true-colour*
2	U16	*máximo de vermelho*
2	U16	*máximo de verde*
2	U16	*máximo de azul*
1	U8	*deslocamento de vermelho*
1	U8	*deslocamento de verde*
1	U8	*deslocamento de azul*
3		*enchimento com zeros (padding)*

O formato natural dos *pixeis* a ser usado será o *formato de pixel do servidor* a menos que o cliente peça um tipo diferente usando a mensagem de *Definir o Formato dos Pixeis*.

Bits por pixel é o numero de bits usado para cada *pixel* transmitido, devendo ser maior ou igual que a *profundidade* que é o numero útil de bits por *pixel*. Nas versões actuais, os *bits por pixel* devem ser 8, 16 ou 32 (menos de 8 bits por *pixel* ainda não são suportados). A *flag big-endian* será diferente de zero se *pixeis* de mais do que um byte são para ser interpretados como *big-endian*.

Se a *flag true-colour* for diferente de zero, então os últimos seis itens especificam como extrair as intensidades de vermelho, verde e azul do *formato de pixel.* O *máximo de vermelho* é o máximo valor de vermelho (= 2^n – 1 onde n é o numero de bits usado para o vermelho). O *deslocamento de vermelho* é o número de deslocamentos necessários para obter o bit menos significativo do valor de vermelho do *pixel*. Se a *flag true-colour* for zero, então o servidor usa valores de *pixeis* que não são directamente compostos por intensidades de vermelho, verde e azul, mas ao invés disso, servem de índices para um mapa de cores. As entradas do mapa de cores são definidas pelo servidor usando a mensagem de *Definir as Entradas do Mapa de Cores*.

A.5.2 Tipos de segurança

A.5.2.1 Nenhum

Não é necessária autenticação e os dados do protocolo são enviados sem serem cifrados.

A.5.2.2 Autenticação do VNC

É usada a autenticação própria do VNC e os dados do protocolo são enviados sem serem cifrados. O servidor envia um "desafio" aleatório de 16-bytes:

Numero de bytes	Tipo	[Valor]	Descrição
16	U8		*desafio*

O cliente cifra o *desafio* com DES, usando a palavra-chave fornecida pelo utilizador e envia o resultado de 16-bytes de volta ao servidor:

Numero de bytes	Tipo	[Valor]	Descrição
16	U8		*resposta*

O protocolo depois prossegue normalmente.

A.5.3 Mensagens do cliente para o servidor

A.5.3.1 Definir o formato dos pixeis

Esta mensagem define o formato em que os valores dos *pixeis* devem ser enviados nas mensagens de *Actualização ao Framebuffer*. Se o cliente não enviar uma mensagem a *Definir o formato dos pixeis*, então o servidor envia os valores dos *pixeis* no seu formato natural conforme foi especificado na mensagem de *Inicialização do servidor.*

Se a *flag de true-colour* estiver a zero, isto indica que o mapa de cores é para ser usado. O servidor pode definir qualquer das entradas do mapa de cores usando a mensagem *Definir as Entradas do Mapa de Cores*. Imediatamente depois de o cliente ter enviado esta mensagem, o mapa de cores fica vazio, mesmo que tenha sido previamente preenchido pelo servidor.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	0	*tipo da mensagem*
3			*enchimento com zeros (padding)*
16	PIXEL_FORMAT		*formato dos pixeis*

A.5.3.2 Definir as Codificações

Esta mensagem indica ao servidor os vários tipos de codificação no qual a informação dos *pixeis* pode ser enviada. A ordem dos tipos de codificação na mensagem é uma dica dada pelo cliente para a sua preferência (a primeira codificação especificada é a preferida). No entanto o servidor pode ou não escolher o tipo de codificação usando esta dica. Os dados dos *pixeis* podem ser sempre enviados no modo de codificação *Raw* mesmo que este modo não seja especificado nesta mensagem.

Em adição às codificações genuínas, o cliente pode sempre pedir pseudo-codificações, para declarar ao servidor que suporta certos tipos de extensões ao protocolo. Um servidor que não suporte a extensão simplesmente ignora-a. Isto faz com que o cliente assuma que o servidor não suporta a extensão até obter uma confirmação específica deste.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	2	*tipo da mensagem*
1			*enchimento com zeros (padding)*
2	U16		*número-de-codificações*

Seguido de *numero-de-codificações* repetições do seguinte:

Numero de bytes	Tipo	[Valor]	Descrição
4	S32		*tipo de codificação*

A.5.3.3 Pedido de Actualização ao Framebuffer

Esta mensagem notifica o servidor de que o cliente está interessado na área do *framebuffer* especificada por *posição-x*, *posição-y, largura* e *altura.* O servidor normalmente responde a um *Pedido de Actualização ao Framebuffer* enviando uma mensagem de *Actualização ao Framebuffer*. Note-se no entanto, que uma única mensagem de *Actualização ao Framebuffer* pode ser enviada como resposta a múltiplos *Pedidos de Actualização ao Framebuffer*.

O servidor assume que o cliente mantém cópias de todas as partes do *framebuffer* nas quais está interessado, o que significa que normalmente o servidor só precisa de enviar actualizações incrementais ao cliente. No entanto, se por alguma razão, o cliente perdeu o conteúdo de alguma área em particular que precisa, então envia ao servidor o *Pedido de Actualização ao Framebuffer* com o campo *incremental* a zero, pedindo assim ao servidor que envie o mais rapidamente possível o conteúdo desta área. Se o cliente não perdeu o conteúdo da área em que está interessado, então envia o campo *incremental* diferente de zero, o que faz com que o servidor envie uma A*ctualização ao Framebuffer,* se e quando ocorrerem mudanças na área especificada (note-se que pode ocorrer um tempo indefinido entre o pedido de actualização e a resposta).

No caso de um cliente rápido, este pode querer regular o ritmo a que envia os *Pedidos de Actualização ao Framebuffer* incrementais, para evitar sobrecarregar a rede.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	3	*tipo de mensagem*
1	U8		*incremental*
2	U16		*posição-x*
2	U16		*posição-y*
2	U16		*largura*
2	U16		*altura*

A.5.3.4 Eventos de Teclado

São considerados *Eventos de Teclado* o pressionar ou libertar de uma tecla, sendo que a *flag pressionada* é diferente de zero se a tecla estiver agora pressionada, e é zero se foi agora libertada. A *tecla* em si é especificada usando os valores de "Keysym" definidos no sistema *X Window* (o que para a maioria das teclas é igual ao seu valor ASCII correspondente).

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	4	*tipo de mensagem*
1	U8		*flag pressionada*
2			*enchimento com zeros (padding)*
4	U32		*tecla*

A.5.3.5 Eventos de Rato

Estes eventos indicam se houve movimento do rato ou se foi pressionada ou libertada alguma tecla do rato. O rato encontra-se na (*posição-x, posição-y*) e o estado dos botões de 1 a 8 são representados pelos bits de 0 a 7 da *mascara de botões* respectivamente, sendo que 0 indica que o botão está descarregado e 1 indica que o botão está pressionado.

Num rato convencional, os botões 1, 2 e 3 correspondem ao botão da esquerda, do meio, e da direita do rato. Se o rato tiver roda de *scroll*, cada passo da roda para cima é representado por um pressionar e libertar do botão 4 e cada passo da roda para baixo é representado por um pressionar e libertar do botão 5.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	5	*tipo de mensagem*
1	U8		*mascara de botões*
2	U16		*posição-x*
2	U16		*posição-y*

A.5.3.6 Texto enviado pelo cliente

Actualmente a única forma de transferir texto é através do formato Latin-1 (ISO 8859-1). Os fins de linha são representados por um '\n' (*linefeed)* e não são precisos '\r' (*carriage-return*).

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	6	*tipo de mensagem*
3			*enchimento com zeros (padding)*
4	U32		*comprimento*
comprimento	Vector de U8		*texto*

A.5.4 Mensagens do servidor para o cliente

A.5.4.1 Actualização ao Framebuffer

Uma *Actualização ao framebuffer* consiste numa sequência de rectângulos com dados de *pixeis* que o cliente deve pôr no seu *framebuffer*. É enviada em resposta a um *Pedido de Actualização ao Framebuffer* por parte do cliente, podendo existir um período indefinido entre pedido e resposta.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	0	*tipo da mensagem*
1			*enchimento com zeros (padding)*
2	U16		*número-de-rectângulos*

Esta mensagem inicial é seguida por *número-de-rectângulos* rectângulos com dados de *pixeis*. Cada rectângulo consiste em:

Numero de bytes	Tipo	[Valor]	Descrição
2	U16		*posição-x*
2	U16		*posição-y*
2	U16		*largura*
2	U16		*altura*
4	S32		*tipo de codificação*

Seguido pelos dados de *pixeis* no formato especificado.

A.5.4.2 Definir as Entradas do Mapa de Cores

Quando o formato dos *pixeis* usa um 'mapa de cores', esta mensagem diz ao cliente que os valores de *pixeis* especificados devem ser mapeados nas intensidades *RGB* (*Red, Green and Blue*) fornecidas.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	1	*tipo de mensagem*
1			*enchimento com zeros (padding)*
2	U16		*primeira cor*
2	U16		*número-de-cores*

Seguido por *número-de-cores* repetições do seguinte:

Numero de bytes	Tipo	[Valor]	Descrição
2	U16		*vermelho*
2	U16		*verde*
2	U16		*azul*

A.5.4.3 Som (beep)

Reproduz um som (*beep*) no cliente se este tiver capacidade para tal.

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	2	*tipo de mensagem*

A.5.4.4 Texto Enviado pelo Servidor

Actualmente a única forma de transferir texto é através do formato Latin-1 (ISO 8859-1). Os fins de linha são representados por um '\n' (*linefeed*)e não são precisos '\r' (*carriage-return*).

Numero de bytes	Tipo	[Valor]	Descrição
1	U8	3	*tipo de mensagem*
3			*enchimento com zeros (padding)*
4	U32		*comprimento*
comprimento	Vector de U8		*texto*

A.5.5 Codificações

As codificações definidas neste documento são:

Numero	**Nome**
0	Raw
1	CopyRect
2	RRE
-239	Pseudo-Codificação do Cursor
-223	Pseudo-Codificação do Tamanho do Desktop

Outras codificações registadas são:

Numero	**Nome**
4	*CoRRE*
5	*Hextile*
6, 7, 8	*zlib, tight, zlibhex*
-256 até -240	
-238 até -224	
-222 até -1	Opcões do *TightVNC*

A.5.5.1 Codificação Raw (sem codificação)

O tipo de codificação mais simples é aquele em que se envia os *pixeis* sem serem codificados. Neste caso a informação contida nos *pixeis* consiste em *largura* x *altura* valores de *pixeis* (onde a *largura* e a *altura* são as dimensões do rectângulo). Os valores representam simplesmente cada *pixel* num varrimento em linha da esquerda para a direita. Todos os clientes devem suportar este tipo de codificação (ou ausência dela) e os servidores devem apenas produzir dados nesta codificação a menos que o cliente peça especificamente outro tipo de codificação.

Numero de bytes	Tipo [Valor]	Descrição
largura x *altura* x *bytesporpixel*	Vector de PIXEL	*pixeis*

A.5.5.2 Codificação CopyRect

A codificação *CopyRect* (copia rectângulo) é simples e eficiente e pode ser usada quando o cliente já tem os mesmos dados de *pixel* noutra parte do seu *framebuffer*. A codificação que é enviada consiste simplesmente nas coordenadas x, y de onde o cliente pode copiar o rectângulo de *pixeis* do seu *framebuffer*. Isto pode ser usado nas mais variadas situações, sendo uma das mais óbvias quando por exemplo o utilizador arrasta uma janela pelo ecrã. Uma razão menos óbvia é por exemplo para optimizar o desenhar no ecrã de texto ou de outros padrões repetitivos.

Numero de bytes	Tipo [Valor]	Descrição
2	U16	*posição-x-origem*
2	U16	*posição-y-origem*

A.5.5.3 Codificação RRE

Os rectângulos codificados neste formato, chegam ao cliente numa forma que podem ser desenhados imediatamente e eficientemente pelo mais simples dos motores gráficos. Esta codificação não é apropriada para ambientes de trabalho complexos mas pode ser útil em algumas situações.

O conceito por detrás da codificação RRE é a de dividir um rectângulo de *pixeis* em sub-regiões (sub rectângulos) cada uma composta por *pixeis* de um único valor, sendo que a união destas compõe o rectângulo original. A partição quase-óptima de um rectângulo em sub rectângulos é relativamente fácil de calcular.

A codificação consiste num *pixel* de fundo, V_b (tipicamente o valor do *pixel* que mais se repete) e do número de sub rectângulos, seguido da listagem destes. Os sub rectângulos consistem no tuplo <v, x, y, w, h> onde v ($\neq V_b$) é o valor do pixel, (x, y) são as coordenadas do sub rectângulo relativamente ao canto superior esquerdo do rectângulo e (w, h) são a largura e altura do sub rectângulo. O cliente pode assim desenhar o rectângulo original enchendo-o com o *pixel* de fundo e depois desenhar rectângulos menores correspondendo a cada sub rectângulo.

Na rede, os dados começam com o cabeçalho:

Numero de bytes	Tipo [Valor]	Descrição
4	U32	*número-de-subrectangulos*
bytesPorPixel	PIXEL	*valor do pixel de fundo*

E é seguido por *numero-de-subrectangulos* repetições da seguinte estrutura:

Numero de bytes	Tipo [Valor]	Descrição
bytesPorPixel	PIXEL	valor do pixel do sub rectângulo
2	U16	*posição-x*
2	U16	*posição-y*
2	U16	*largura*
2	U16	*altura*

A.5.6 Pseudo-Codificações

A.5.6.1 Pseudo-Codificação do cursor

Um cliente que peça esta pseudo-codificação está a declarar que é capaz de desenhar o cursor do rato localmente. Isto pode melhorar significativamente a performance sobre ligações lentas. O servidor define a forma do cursor enviando um pseudo-rectângulo com a pseudo-codificação do cursor, como parte de uma actualização. A *posição-x* e a *posição-y* do pseudo-rectângulo indicam a ponta do cursor, e a *largura* e a *altura* indicam a largura e a altura respectivas do cursor em *pixeis*. Os dados desta mensagem consistem em *largura* x *altura* valores de *pixeis* seguidos por uma *máscara*. A *máscara* consiste num varrimento de linhas da esquerda para a direita e de cima para baixo, onde cada linha é preenchida com zeros (*padded*) até perfazer um número inteiro de bytes *floor((largura + 7)/8) + altura)*. Dentro de cada byte, o bit mais significativo representa o *pixel* mais à esquerda, sendo que o bit 1 significa que o *pixel* correspondente do cursor é válido.

Numero de bytes	Tipo [Valor]	Descrição
largura x *altura* x *bytesporpixel*	Vector de PIXEL	*pixeis-do-cursor*
floor((largura + 7)/8) + altura)	Vector de U8	*mascara*

A.5.6.2 Pseudo-Codificação do Tamanho do Desktop

Um cliente que peça esta pseudo-codificação está a declarar que é capaz de aceitar uma alteração à largura e/ou altura do *framebuffer*. O servidor altera o tamanho do ambiente de trabalho enviando um pseudo-rectângulo com a pseudo-codificação de *Tamanho do Desktop* como o ultimo rectângulo de uma actualização. A *posição-x* e a *posição-y* do pseudo-rectângulo são ignoradas e a *largura* e a *altura* indicam as novas dimensões do *framebuffer*. Mais nenhuma informação é associada a este pseudo-rectângulo.

Anexo B. Manual de Utilização (User Guide)

USER GUIDE

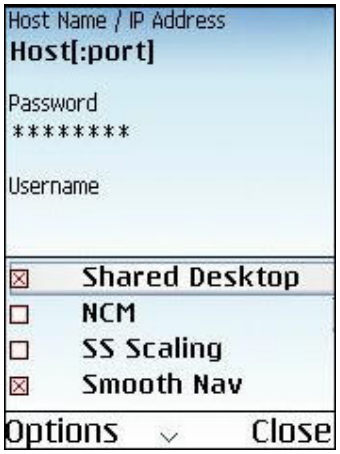
J2ME VNC by Michael Lloyd Lee, modified by Nuno Freire

WinVNC Version 3.3.3 r7, modified by Nuno Freire

B.1 Viewer

The J2ME VNC original version made by Michael Lloyd Lee has been slightly changed (improved I hope) in this release. It now supports some new features like server side scaling, sound transfer and user profiles, requiring a modified VNC server (which is distributed with this version of the software).

The initial form looks like the following:



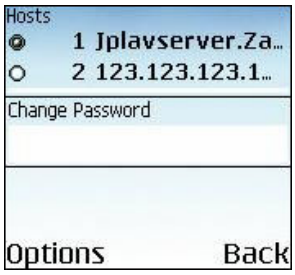
In the left figure you can see some text boxes where you can enter the host's address to which you wish to connect, the password to authenticate the viewer and a username. The content of this last text box is only taken into account if you're using the VNC server distributed with this version of the software which supports a new functionality (user profiles). On the lower bottom, you can see some check boxes which are:

- **Shared Desktop** – If you want to share the server desktop with other possible viewers or not.
- **NCM** (Nokia Compatibility Mode) – If you're having problems with your old Nokia phone, perhaps you should try to activate this mode.
- **SS** (Server Side) **Scaling** – This mode only works if you're using the VNC server distributed with this version of the software. It does what the name suggests.
- **Smooth Navigation** – A new mode which allows for a smoother navigation through the desktops in exchange for more available memory on the mobile platforms.

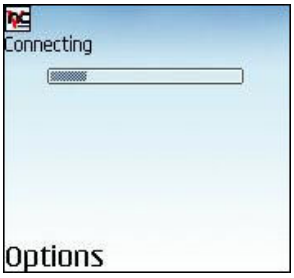
In the right figure you can see the options accessible from the initial form:

- The **Log** option lets you see the log saved during the last connection made.
- The **Connect** option lets you connect to the selected host using the corresponding password. It requires that the hostname text box is filled.
- The **Add Host** option puts the value of the hostname text box in the **address book**
- The **Address book** option moves you to a form (shown next) where you can see the previously recorded hosts, change the passwords associated with them, delete them and

select the one you wish to connect to. You can also set a master password to limit the access to this form to authorized users.



After choosing the **Connect** option you will be transported to the following form, where you can see a progress bar indicating how the connection is occurring.



In this form you can't do much except waiting for the connection to be established, choose to **Exit** the application or choose to see the **Log** option.

After this screen, the session to the VNC server is established and the user can now view and control the remote desktop. The input system remains similar to the original with a few keys added:

- Nav mode: '1' = call mouse
  '3' = zoom in
  '9' = zoom out
  Direction buttons (and 2,4,6,8) move the screen
  '*' = change mode
- Mouse mode: '1' = call mouse
  '3' = scroll up
  '5' = click
  '9' = scroll down
  '7' = press and hold left button/release left button
  '0' = right click
  '#' = double click
  'call key' = enter
  'cancel key' = backspace
  Direction buttons (and 2,4,6,8) move the mouse[4]
  '*' = change mode

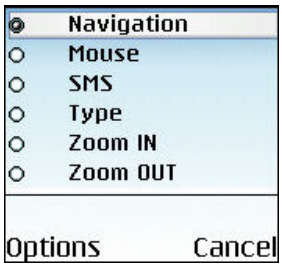
[4] By moving the mouse outside the screen, you can now also navigate through the desktop using this mode

- SMS mode: ‘call key’ = enter
  ‘cancel key’ = backspace
  ‘*’ = change mode
  (To change between lower and upper case a new option, **Case**, appears in the main menu)

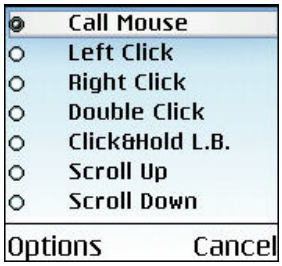
- Type mode: ‘call key’ = enter
  ‘cancel key’ = backspace
  ‘*’ = change mode

The options accessible when the connection is established are the following:

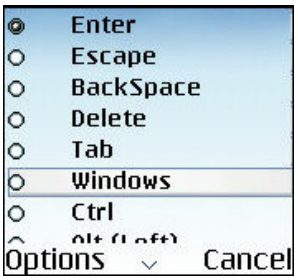
- The **Change Mode** option lets you select the desired mode and allows you to zoom in or zoom out the display. You can also change modes using the ‘*’ key and zoom in and out using the ‘3’ and ‘9’ keys while in the navigation mode.



- The **Mouse** option gives you a list of all the mouse events you can use. *Call mouse* sends the pointer to the center of the screen. A first selection of *Click&Hold L.B.* presses the left mouse button and leave it that way and a second selection of the same event un-presses the left button. You can use this, for instance, to select a phrase in a text or to move a window across the desktop. The other events available are self explanatory.

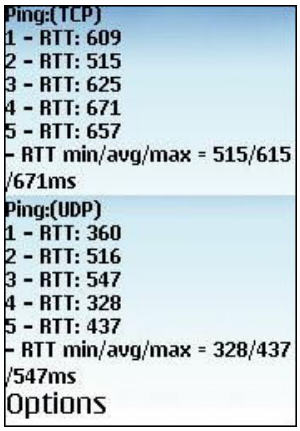


- The **Special Keys** option lets you see a list of keys that are not accessible from most of the mobile phones keyboards. The list is not exhaustive but the main keys are present. Combinations of keys can be made, for instance, you could press 'Ctrl' (which remains pressed), then press 'Shift' (which also remains pressed) and then press 'Escape'. In alternative you can create a macro in the About Midlet that does the before-mentioned action. You can also use combinations like pressing 'Ctrl' and then typing an 'a' (must be lower case) in SMS mode to obtain the windows "select-all" shortcut.



- The **Stats** option lets you see some stats obtained during the connection, like, throughput, bytes transferred, memory used by the application…This may not be of much interest to the regular user.

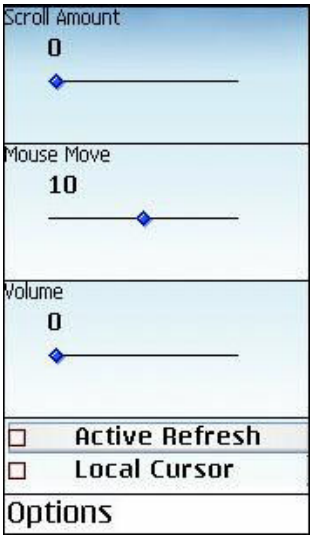
- The **Ping** option lets you make a ping to the server in an attempt to estimate the round trip time. It uses two methods and may also not be of much interest to the regular user. Both the Ping and the Stats options may be removed in a future version.



- The **Refresh** option sends an update request to the server for the whole desktop.

- The **Options** option lets you configure a bunch of variables in the viewer. The 'Scroll Amount' and the 'Mouse Move' amount are self explanatory. The 'Active Refresh' indicates whether or not you want the screen to be refreshed every couple of seconds. The 'Local Cursor' indicates whether or not every movement of the cursor in the viewer

should reflect an action by the server or if only the clicks should be sent to the server. The 'Volume' is a new feature only available if you're using the VNC server version distributed with this software, and a value of zero represents that no sound transfer session has been initiated and a value higher then zero indicates the volume at which the sound should be played in the mobile phone.



- The **Enter Text** option allows the user to send text to the server. It also allows **past**ing to the form a previously copied text from the server (available in it's clipboard), changing it and sending it back. If the user uses the **Copy** option in this form, all the user text will be available in the server clipboard.

External to the main application there's an About MIDlet where you can define Macros using a special format. For instance, if you want to send "Hello" to the server, the macro should be defined as "`!H!e!l!l!o`" and if you want to send ctrl alt delete, it should be defined as "`<\c<\a!\d>\a>\c`". All the user defined macros will be accessible through the **Special Keys** form.

The letter combos available are the following:

'`x`' – literal,
'`\d`' – Delete,
'`\a`' – Alt,
'`\m`' – Meta,
'`\c`' – Control,
'`\s`' – Shift,
'`\e`' – Escape,
'`\n`' – newline,
'`\1`' - F1
...
'`\9`' – F9
'`\0`' – F10
'`\A`' - F11
'`\B`' - F12
'`\\`' – \
'`\p`' – |

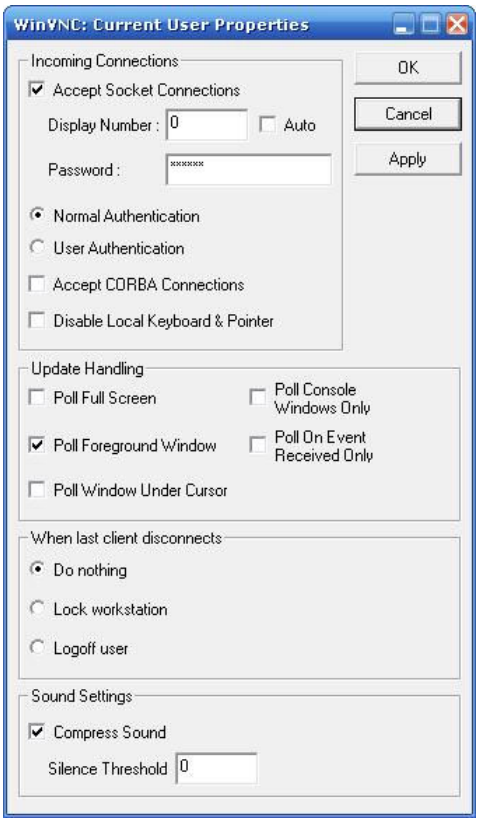
And the commands available are the following:

'`<`' - key down
'`>`' - key up
'`!`' - key down then up (ie press)
'`mXXXX,XXXX`' - Mouse Move To, must be four digits
'`c`' - Click (primary mouse button)
'`MXXXX,XXXX`' - Move Screen To, follows the same standard as 'm'

B.2 Server

The corresponding “enhanced” VNC server was the only one found on the Internet that had server side scaling extensions and was originally distributed with the PalmVNC software. Besides the server side scaling extensions, this software was modified to support sound transfer and user profiles. It also has some new features that you may recognize from newer versions of the VNC server (like RealVNC). To configure the new options you simply have to right-click on the VNC server icon on the taskbar.

The **Properties** menu now let’s you choose from 2 different security types. The **Normal Authentication** is the standard for any VNC server application and you may want to use this if you’re using a regular VNC viewer. The **User Authentication** is an extension to the protocol which lets you use the new “user profile” features. This new feature allows the creation of user profiles in the server and when a viewer connects using a username that matches any of the previously configured profiles, a special application may be run and the viewer will only have access to it (instead of the whole desktop) and a few more properties may be configured for each profile. If no match is made the **standard** user profile will be used.

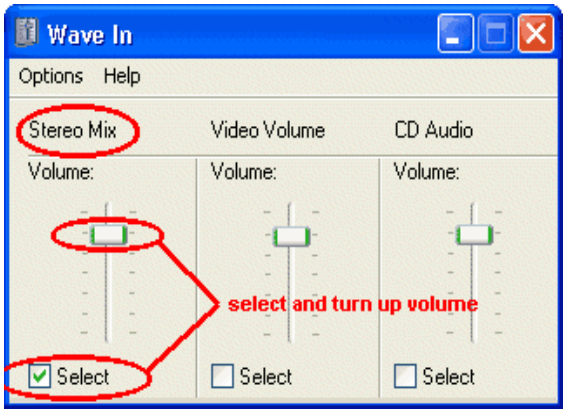


Another new functionality is sound support. This lets the viewer listen to whatever is playing on the server or being captured by the microphone, depending on how you configure the windows recording settings. You may choose to **compress** the transferred **sound** or not and you may choose a **silence threshold** if you want to have some sort of silence suppression. The silence suppression only works if you choose to compress the transferred sound and a good value for it may be ‘10000’ but you may want to play with this number. A value of zero means that no sound suppression will be made and a value of ‘32000’ or higher means that every sound will be suppressed and therefore, no sound will be transferred to the viewer.

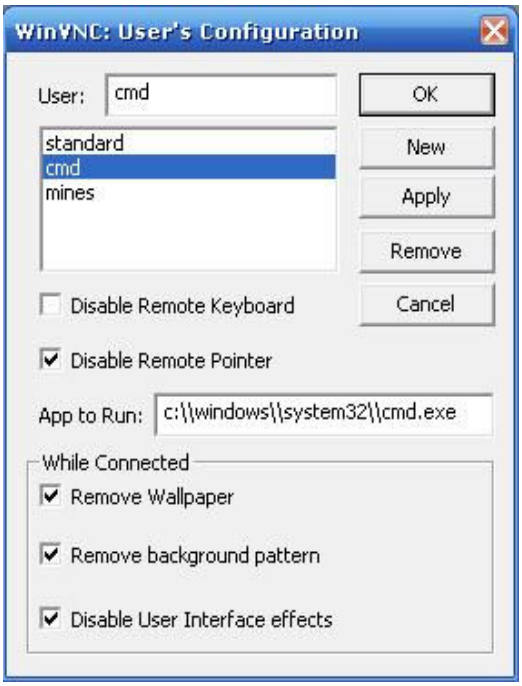
To choose between capturing the audio from the microphone and capturing the audio that is being played internally on the computer you may want to go the Wave In window (Recording section) and select the appropriate channel. Typically, only one live channel is live, but once you find it, it always works. Unfortunately, each audio card uses different channel names, so you must look for a channel named something like this:

Stereo Mix, Record Master, What U Hear, Sum Balance, Stereo Mixer, Mixed Output, Wave Out Mix, Wave/MIDI/CC…

To record external audio, use the Microphone channel.



To create, change and/or delete user profiles you need to access the **Users** menu. There you may differentiate the users by selecting different properties for each one. You may also want to restrict the user control to a specific application. For that, you need to enter the full location of the application you want to run when the client connects, substituting all the '\' by '\\'. At the time, this new feature only allows one client to be connected at one given moment or the viewer will became all mixed up.



If you're behind a router or a firewall you need to open 3 ports to be able to access all the new functionalities. If you have the **Display Number** set to zero, this means that the port used by the VNC to communicate is the 5900, the port used to accept a sound transfer request from the viewer is the 5901 and the port used to be able to ping the server is the 5902. You may not allow access to the last two mentioned ports but this way you won't have access to the corresponding functionalities.

For other information about the VNC server please refer to the original (WinVNC) manual.

Anexo C. Algoritmos de compressão e descompressão

C.1 Compressão

```
Char codecRep[4]; //tomar conta dos símbolos repetidos
Char bufOut[tamanho_do_pacotes_de_som]; //buffer de saída que conterá o
                                        pacote comprimido
Short numOfRep;   //indica o número de repetições (-1) do simbolo actual
Int currentPos;   //indica a posição actual no buffer de saída
Bool hotSymbol;   //Indica se foi encontrado um símbolo '<'

codecRep[0] = bufOut[0] = Som[0];
numOfRep=0;
currentPos = 0;

SE Som[0] = '<' ENTAO { //Som é o buffer que contém o áudio a comprimir
    hotSymbol = verdadeiro;
    bufOut[1] = Som[0];
    currentPos = 2;
}
SENAO
    hotSymbol = falso;

PARA i=0 ATE tamanho_do_pacote_de_som {
    SE currentPos >= tamanho_do_pacotes_de_som ENTAO sai do ciclo;
    SE Som[i] == '<' ENTAO hotSymbol = verdadeiro;
    //O símbolo actual é igual ao anterior mas ainda não foi repetido 4x
    SE Som[i] == codecRep[numOfRep] E numOfRep<3 E hotSymbol = falso ENTAO {
           codecRep[++numOfRep] = Som[i];
           bufOut[currentPos+numOfRep] = Som[i];
    }
    //O símbolo actual é igual ao anterior e é a 5ª repetição deste
    //ou é um '<' que precisa de ser codificado
    SE (Som[i] == codecRep[numOfRep] E numOfRep==3) OU (hotSymbol = falso E
    codecRep[0] != '<') ENTAO {
           SE hotSymbol = verdadeiro ENTAO{
                  codecRep[0] = Som[i];
                  currentPos += numOfRep+1;
                  numOfRep = 0;
           }
           SENAO numOfRep ++;
           bufOut[currentPos++] = '<';
           bufOut[currentPos++] = Som[i];

    }
    //O símbolo actual é igual ao anterior e o número de repetições é
      superior a 4 por isso vai contando o número de repetições
    SE Som[i] == codecRep[0] E  (numOfRep>3 OU hotSymbol = verdadeiro) ENTAO
           numOfRep ++;
    //O símbolo actual é diferente do anterior e houve 5 ou mais símbolos
      repetidos
    SE Som[i] != codecRep[0] E (numOfRep>3 OU hotSymbol = verdadeiro) ENTAO{
           numOfRep ++;
           bufOut[currentPos] = numOfRep //copia dois bytes (short)
           currentPos += 2;
           codecRep[0] = Som[i];
           bufOut[currentPos] = Som[i];
           numOfRep = 0;
           hotSymbol = falso;
    }
    //O símbolo actual é diferente do anterior e não houve 5 ou mais
      símbolos repetidos
    SENAO {
           codecRep[0] = Som[i];
           currentPos += numOfRep+1;
           bufOut[currentPos] = Som[i];
           numOfRep = 0;
    }
}
//Terminou o pacote de som a comprimir, agora pode haver ou não uma ultima
  compressão a fazer, conforme os últimos símbolos tenham sido repetidos ou não.
SE numOfRep > 3 ENTAO{
    numOfRep++;
    bufOut[currentPos] = numOfRep; //copia dois bytes (short)
    currentPos += 2;
}
SENAO currentPos += numOfRep+1;
```

C.2 Descompressão

```
SE tamanho_do_pacote_recebido != tamanho_do_buffer ENTAO {//se vier comprimido
    Int currentPos = 0;
    Int numOfRep = 0;
    PARA i=44 ATE tamanho_do_pacote_recebido { //começa em 44 que é depois do
                                                Cabeçalho
          SE Som[i] == '<' ENTAO {
                numOfRep = Som[i+2] //copia dois bytes (short)
                PARA j=0 ATE numOfRep { //copia o simbolo em i+1 numOfRep
                                          vezes
                SE currentPos < tamanho_do_buffer ENTAO
                      Buffer_temporario[currentPos++] = Som[i+1];
                i +=3;
          }
          SENAO {
                SE currentPos < tamanho_do_buffer ENTAO
                      Buffer_temporario[currentPos++] = Som[i];

          }
    }
    //Som é o buffer que vai ser usado pelos players. Os primeiros 44 bytes são o
      cabeçalho WAVE e o restante será áudio
    Copia buffer_temporario PARA Som
}
```

Anexo D. Repetições de símbolos num ficheiro de áudio