Acesso Remoto à computadores

André B. Oliveira, César H. Kallas, Marcelo G. Hyppolito, Rafael B. Curi

CEATEC – Pontifícia Universidade Católica (PUC)
Campinas – RS – Brazil

Centro de Ciências Exatas, Ambientais e Tecnológicas

cesarkalals at gmx.net rafabcuri@yahoo.com.br mghyppolito@gmail.com

Resumo. *O objetivo das tecnologias de acesso remoto a computadores é prover um ambiente de trabalho remoto aos usuários. Envolve um servidor de terminais e vários terminais que o acessam e realizam operações através das entradas padrão, que são encaminhadas do terminal para o servidor pelo sistema. Os resultados das ações exibidos na interface de saída do servidor é – da mesma forma – encaminhada ao terminal de acesso para visualização do usuário.*

O presente trabalho é um estudo sobre as principais tecnologias de acesso remoto a computadores. A seguir, os principais aplicativos que implementam um protocolo de controle remoto são descritos. São exibidos e explicados as arquiteturas dos sistemas, de seus usos, além dos seus protocolos de comunicação.

1. SSH – Secure Shell

1.1 Nome

SSH - Secure Shell

1.2 Descrição

O SSH é um programa e um protocolo de rede que permite administrar máquinas remotamente, executando inclusive apliativos gráficos, permite transferir arquivos de várias formas diferentes e permite também encapsular outros protocolos, possibilitando, por exemplo, acessar uma seção VNCatravés de um tunel seguro.

1.3 Autor/Empresa

O SSH foi desenvolvido pela empresa SSH Communications Security, criada em 1995 para desenvolver soluções seguras de comunicação.

1.4 Site

O site do SSH, onde é possível copiar o pacote, é http://www.ssh.fi.

1.5 Função Resumida

O SSH possibilita o acesso remoto via linha de comando por usuários que tenham o login e senha de uma das contas do sistema. Este é um acesso completo via terminal (rede ou internet), limitado aos privilégios do login do usuário.

Sua principal vantagem sobre outros protocolos, como por exemplo o telnet é a segurança, o SSH utiliza criptografia e protocolo de autenticação mais seguros.

1.6 Funcionamento

1.6.1 Arquitetura

O pacote SSH consistem de vários programas que auxiliam na segurança do protocolo:

1.SSHD

Servidor de conexões do protocolo SSH, deve ser executado como root e normalmente é instalado nos arquivos de inicialização das estações (rc files) mas é possível também iniciá-lo via o inetd. Como o servidor precisa calcular um par de chave publica/chave pivada de 1024 bits a cada inicialização, o método do inetd não é recomendado para máquinas de baixo desempenho. O servidor escuta a porta 22.

2.SSH

Substituto do RSH com algumas melhorias. Por exemplo, se não houver uma entrada nos arquivos .rhosts, ou .shosts, para a máquina cliente, o servidor tenta autenticar pelo servidor de chavers RSA ou pela senha. Existe uma pequena mudança de sinaxe pra os comandos remotos trocando o nome do usuário.

3.SLOGIN

Semelhante ao rlogin e assim como o rsh recai no rlogin quando nenhum comando é dado, o ssh também recai num shell criptografado quando executado sem um comando explícito para a máquina remota.

4.SCP

Semelhante aos rpc, com a vantagem de não derrubar a conexão se a etação remota não possuir uma referência à máquina local nos seus arquivos hosts.equiv e .rhosts. Caso não obtenha autorização pelo .rhosts, a transferência pode ser autorizada por RSA pessoal ou password transmitido codificado.

5.SSH-KEYGEN

Gerador de chaves RSA. Cada usuário pode ter as suas próprias chaves RSA, que são geradas e colcadas no diretório .ssh. Estas chaves são utilizadas para a autenticação das conexões por chaves RSA. As chaves são geradas baseadas numa frase senha, o que aumenta a sua segurança em relação ao sistema de senha das estações ue em muitos casos é limitada a poucos caracteres. O arquivo de senha gerado, ssh/identity, deve terpermissão de leitura/escrita apena para o usuário.

6.SSH-AGENT

Espécie de servidor de chaves RSA para uma seção, normalmente deve ser executado a partir do arquivo .login nas máquinas com login textual e .xsession nas máquinas com login controlado por xdm e seus equivalentes.

6.SFTP

Cliente FTP com suporte a comunicação segura.

6.SFTP-SERVER

Servidor FTP com suporte a comunicação segura.

6.SSH-ADD

Adiciona chaves de autentificação DSA ou RSA ao programa de autentificação.

6.SSH-COPY-ID

Usado para instalação do arquivo identity.pub em uma máquina remota.

1.7 Protocolo

SSH é um protocolo para login remoto seguro e outros serviços contextualizados em redes abertas. Esse protocolo é dividido em três componentes básicos:

· Camada de Transporte: Provê sigilo, integridade e autenticação do servidor. Opcionalmente, pode ser oferecido um serviço de compressão de dados. Essa camada geralmente é rodada em cima de uma conexão TCP/IP, mas é possivel que ela seja usada em cima de qualquer outro tipo de protocolo

confiável e orientado a conexão.

· Protocolo de Autenticação: Autentica o usuário perante o servidor. Esse protocolo necessita da camada de transporte para o seu funcionamento.

· Protocolo de Conexão: Permite a multiplexação da conexão entre vários canais lógicos. Esse protocolo depende do protocolo de autenticação. Nessa camada encontramos recursos como shell interativo, tunelamento de portas TCP/IP e conexões X11.

O SSH é bastante flexível no sentido de que os algoritmos usados em uma conexão são negociados entre o cliente e o servidor. Isso permite que algoritmos fracos sejam removidos ou novos algoritmos sejam testados sem que haja uma mudança no protocolo.

1.8 Camada de Transporte

A camada de tranporte do SSH é um protocolo seguro de baixo nivel que fornece encriptação, integridade e autenticação do servidor. Observe que aqui somente é feita a autenticação do servidor, sendo que a autenticação do usuario é responsabilidade do protocolo de autenticação.

Os pacotes que são recebidos e enviados por essa camada possuem o seguinte formato:

Tamanho (4 bytes)
Tamanho do preenchimento (1 byte)
Conteúdo (n1 bytes)
Preenchimento (n2 bytes)
MAC (m bytes)

Tamanho: Indica o comprimento do pacote sem incluir o MAC e o próprio campo Tamanho. Tamanho = 1 + n1 + n2.

Tamanho do preenchimento = “n2”: Número de bytes que serão colocados no pacote para que o tamanho total (sem o MAC) seja múltiplo de 8 (ou do tamanho do bloco usado na encriptacao). n2 não pode ser menor que 4.

Conteúdo: Conteúdo útil do pacote. Se algum algoritmo de compressão estiver sendo usado, esse campo é compactado.

Preenchimento: “n2” bytes aleatórios.

MAC: Código de autenticação de mensagem. O número “m” de bytes varia de acordo com o algoritmo que está sendo utilizado. O cálculo do MAC é: mac = MAC(key, nº_do_pacote || pacote_não_encriptado), onde nº_do_pacote é um contador de pacotes enviados (ou recebidos).

1.9 Protocolo de Troca de Chaves

- Troca de strings de versão: Após a conexão ser estabelecida, cliente e servidor trocam strings no formato: "SSH-protoversion-software version comentários\r\n". Essa string é chamada V_S para a string enviada pelo servidor e V_C para a do cliente. A partir de então, todos os dados são transmitidos no formato do pacote citado no item anterior.

1- Negociação dos algoritmos usados: Ambos os lados enviam o pacote SSH_MSG_KEXINIT a seguir:

byte	SSH_MSG_KEXINIT
byte[16]	cookie (bytes aleatórios)
string *	Algoritmos de troca de chaves
string *	Algoritmos de chave pública
string *	Algoritmos de encriptação cliente-servidor
string *	Algoritmos de encriptação servidor-cliente
string *	Algoritmos MAC cliente-servidor
string *	Algoritmos MAC servidor-cliente
string *	Algoritmos compressão cliente-servidor
string *	Algoritmos compressão servidor-cliente
string	Linguagens cliente-servidor
string	Linguagens servidor-cliente
boolean	first_kex_packet_follows
uint32	0 (reservado)

Os campos em destaque (*) são as strings com os nomes dos algoritmos em ordem de preferência. Após os dois lados enviarem esse pacote, a camada de transporte decide qual o algoritmo que será usado no protocolo em cada um dos campos. O algoritmo usado é o primeiro que o cliente possuir que o servidor suporte.

2. Protocolo de troca de chaves. A saída dessa etapa é um segredo compartilhado K e o resultado H de um hash. Esse hash é usado como identificador de secao e como parte dos dados assinados pelo servidor para ele se autenticar perante o cliente.

O protocolo de troca de chaves padrão do SSH é o diffie-hellman-group1-sha1 explicado a seguir:

2.1 Cliente:

2.1.1 gera X randômico

2.1.2 envia

byte	SSH_MSG_KEXDH_INIT
mpint	e = (g^x mod p)

Onde g=2 e p o seguinte primo(120 bits):

FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1
29024E088A67CC74020BBEA63B139B22514A08798E3404DD

EF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245
E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED
EE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381

2.2. Servidor:
2.2.1 gera Y randômico
2.2.2 calcula:
· f = g y mod p
· K = e y mod p
· H = SHA-1(V_C || V_S || I_C || I_S || K_S || e || f || K)
Onde V_C, V_S são as strings de versão; I_C e I_S os conteúdos dos pacotes SSH_MSG_KEXINIT; e K_S a chave pública do servidor.
2.2.3 gera S = assinatura sobre H usando a chave privada
2.2.4 envia:

byte	SSH_MSG_KEXDH_REPLY
string	K_S (chave publica)
mpint	f
string	S – Assinatura sobre H

2.3 Cliente:

2.3.1 Verifica se K_S é realmente a chave pública do servidor, podendo ser feito uma busca em um histórico de conexões. Caso esse passo não seja executado corretamente, o protocolo está sujeito ao ataque man-in-the-middle.
2.3.2 Calcula:
· K = f k mod p
· H = SHA-1(V_C || V_S || I_C || I_S || K_S || e || f || K)
2.3.2 Verifica S com K_S
3. Efetivação do uso de novas chaves com os algoritmos negociados.
Cada lado da conexão envia a seguinte mensagem

byte	SSH_MSG_NEWKEY

A partir de então, todos os pacotes enviados são encriptados de acordo com o algoritmo negociado.

1.10 Reexecução da Troca de Chaves

É recomendável que as chaves de sessão sejam trocadas a cada 1Gb de dados transmitidos ou de hora em hora. Para efetuar essa troca basta que um dos lados mande um pacote do tipo SSH_MSG_KEXINIT. Isso força a execução dos passos 1 ao 3 da troca de chaves. O Identificador da sessão é o “H” da primeira troca de chave, e ele mantêm o seu valor até o término da conexão.

1.11 Vetores de Inicialização e Chaves de Seção

Os vetores de inicializacao e chaves de sessao para os algoritmos de encriptação e MACs são criados da seguinte forma:

	cliente - servidor	servidor - cliente
VI	HASH(K || H || "A" || ID)	HASH(K || H || "B" || ID)
Chave de Encriptação	HASH(K || H || "C" || ID)	HASH(K || H || "D" || ID)
Chave do MAC	HASH(K || H || "E" || ID)	HASH(K || H || "F" || ID)

“HASH” aqui é o algoritmo utilizado na troca de chaves negociada. No caso do diffie-hellman-group1-sha1 por exemplo, HASH é o SHA1.

1.12 Requisição de Serviços

Após o protocolo terminar a troca de chaves e receber o pacote SSH_MSG_NEWKEYS, o sistema pode tomar vários rumos, onde o caso mais comum é a autenticação do usuário. Para isso, deve-se requisitar um serviço através do seguinte pacote:

byte	SSH_MSG_SERVICE_REQUEST
string	Nome do serviço

Se o servidor reconhecer esse serviço, ele deve retornar o próximo pacote:

byte	SSH_MSG_SERVICE_ACCEPT
string	Nome do serviço

Se o servidor recusar o pedido, o cliente deve receber SSH_MSG_DISCONNECT e em seguida ser desconectado.

Obs.: Para o caso da autenticação, nome do serviço é “ssh-userauth”.

1.14 Algoritmos

A seguir temos a tabela com os algoritmos padronizados que foram definidos no protocolo de transporte do SSH. Como citado anteriormente, outros algoritmos podem ser acrescentados nessa lista. Basta utilizar nomes diferente dos citados a seguir, pois estes são reservados para o protocolo.

Compressão	none, zlib
Encriptação	3des-cbc,blowfish-cbc, twofish256-cbc,twofish-, twofish192-cbc, twofish128-cbc, aes256-cbc,

	aes192-cbc, aes128-cbc, serpent256-cbc, serpent192-cbc, serpent128-cbc, arcfour, idea-cbc, cast128-cbc, none
MAC	hmac-sha1, hmac-sha1-96, hmac-md5, hmac-md5-96, none
Troca de Chaves	diffie-hellman-group1-sha1
Algoritmos de chave pública	ssh-dss, ssh-rsa, x509v3-sign-rsa, x509v3-sign-dss, spki-sign-rsa, spki-sign-dss, pgp-sign-rsa, pgp-sign-dss

1.15 Protocolo de Autenticação

Aqui são executados os procedimentos relativos a autenticação do usuário. É assumido que a camada de transporte já está provendo integridade e sigilo na comunicação.

Para ocorrer a autenticação, o cliente deve enviar a seguinte mensagem:

byte	SSH_MSG_USERAUTH_REQUEST
string	user name
string	Nome do serviço
string	Nome do método
… extra …	

O nome do serviço especifica qual o serviço que será iniciado após a autenticaçao. Se o serviço não estiver disponível, o servidor irá desconectar o cliente.

O método é o nome do procedimento usado para a autenticação.

O servidor pode responder de duas formas:

Autenticação confirmada:

byte	SSH_MSG_USERAUTH_SUCCESS

Autenticação falhou:

byte	SSH_MSG_USERAUTH_FAILURE
string	Métodos disponíveis
boolean	Sucesso parcial

Onde sucesso parcial indica que o método de autenticação aceitou o usuário, mas o servidor não aceitou essa autenticação.

Um exemplo de método de autenticação é o que utiliza senha. O pacote SSH_MSG_USERAUTH_REQUEST relative a ele é o seguinte:

byte	SSH_MSG_USERAUTH_REQUEST

string	user name
string	Nome do serviço
string	“password”
boolean	FALSE
string	senha

Embora a senha é informada “às claras”, o pacote todo é criptografado (ao contrário do telnet, por exemplo), impedindo assim o vazamento da senha se alguém estiver ouvindo o tráfego.

O servidor também pode responder com SSH_MSG_USERAUTH_PASSWD_CHANGEREQ, informando que a senha do usuário expirou e que ela deve ser trocada. Para isso, o cliente deve enviadar o seguinte pacote:

byte	SSH_MSG_USERAUTH_REQUEST
string	user name
string	Nome do serviço
string	“password”
boolean	TRUE
string	Senha antiga
string	Senha nova

1.16 Protocolo de Comunicação

Esta é aparte que provê execução de commandos remotos, abertura de shells, tunelamento de conexões TCP/IP e de conexões X11. Todos esses canais são multiplexados em um único tunel encriptado.

O protocolo de comunicação tem o seu funcionamento baseado em canais. Cada canal é identificado por um número em cada uma das pontas. Esse número pode ser diferente em cada um dos lados.

Canais possuem fluxo controlado. Os dados são transmitidos através deles até quando o espaço disponível em suas janelas se esgotar. Quando isso ocorrer, o tamanho das janelas deve ser incrementado com o envio de uma mensagem adequada.

Para abrir um canal na conexão, a próxima mensagem deve ser transmitida por um dos lados.

byte	SSH_MSG_CHANNEL_OPEN
string	Tipo do canal
uint32	no do canal remetente
uint32	Tamanho inicial da janela
uint32	Tamanho máximo dos pacotes
… extra	

…	

O outro lado deve responder o pedido com um dos dois pacotes a seguir:

byte	SSH_MSG_CHANNEL_OPEN_CONFIRMATION
uint32	no do canal destinatário
uint32	no do canal remetente
uint32	Tamanho inicial da janela
uint32	Tamanho máximo dos pacotes
… extra …	

byte	SSH_MSG_CHANNEL_OPEN_FAILURE
uint32	no do canal destinatário
uint32	código do motivo
string	Informaçao adicional
string	Tag de lingugagem

Observe que o no do canal destinatário dessas duas ultimas mensagens é o número do canal remetente de quem requisitou a abertura do canal (SSH_MSG_CHANNEL_OPEN).

Para transferir dados pelo canal, utiliza-se a seguinte mensagem:

byte	SSH_MSG_CHANNEL_DATA
uint32	no do canal destinatário
string	Dados

ou

byte	SSH_MSG_CHANNEL_EXTENDED_DATA
uint32	no do canal destinatário
uint32	Tipo dos dados
string	Dados

O tamanho disponível da janela de quem receber os dados deve ser decrementada pelo número de bytes do campo dados, e ao se esgotar a janela a mensagem a seguir deve ser enviada pelo destinatário:

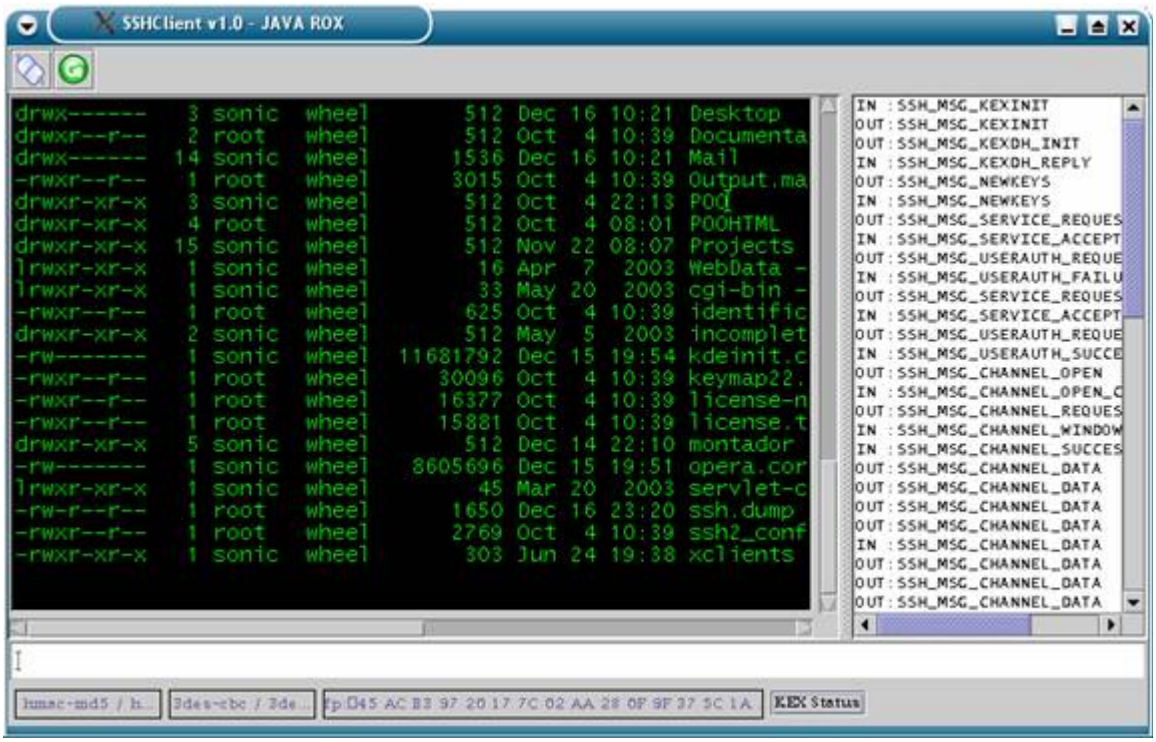
byte	SSH_MSG_CHANNEL_WINDOW_ADJUST
uint32	no do canal destinatário
uint32	no de bytes incremental

Com isso a janela disponibiliza o número de bytes solicitado.

1.17 Implementação do Cliente

A seguir, foi implementado um protótipo de um cliente que se comunica com um servidor SSH. Este cliente possui os seguintes recursos:

· Algoritmo de encriptação: 3des-cbc, none
· Algoritmos MAC: hmac-md5, hmac-sha1
· Algoritmo de troca de chaves: diffie-hellman-group1-sha1
· Algoritmo de chave pública: ssh-dss
· Protocolo de autenticação: password
· Tipo de serviço suportado: shell remoto
· Visualização de pacotes
· Botão de reexecutar troca de chaves



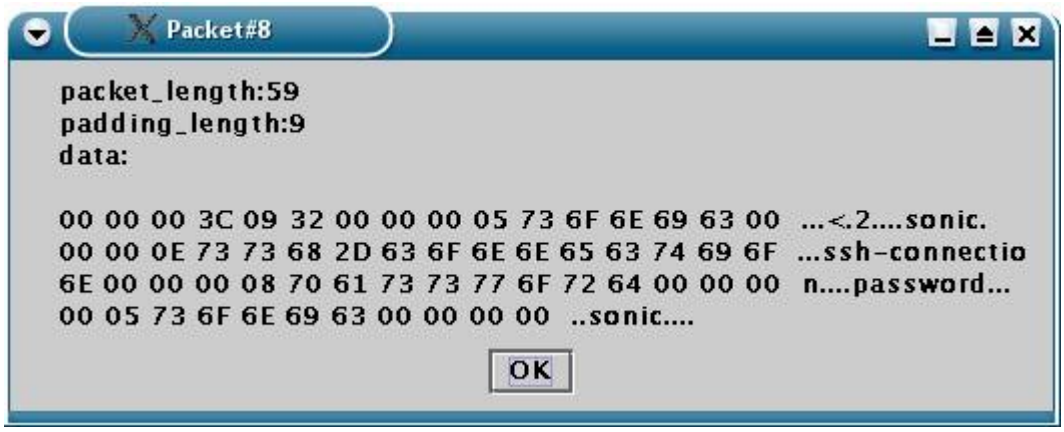
Interface Principal

O 1º botão conecta ou disconecta em um servidor.

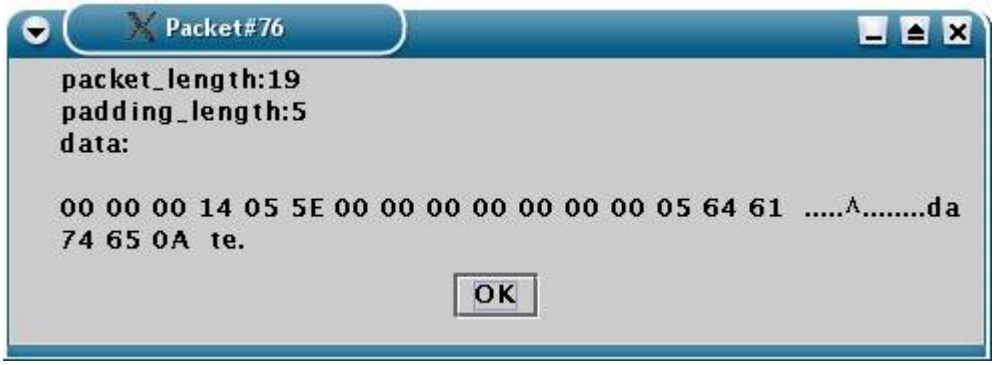
O 2º botão (reexecutar troca de chaves) funciona da seguinte forma: Ao clicá-lo, novas chaves são estabelecidas. Como uma funcionalidade didática, os algoritmos de encriptação (3des-cbc e none) são alternados em cada nova troca, assim como os algoritmos de MAC (hmac-md5, hmac-sha1). Se o servidor não suportar o hmac-md5 ou o none, o cliente não funcionará. Logo, não use o botão de reexecutar troca de chaves se nao se conhecer os algoritmos suportados pelo servidor.

A listagem que aparece no lado direito são os pacotes que entram e saem do aplicativo. Ao dar um duplo clique é possível visualizar o conteúdo de cada pacote.

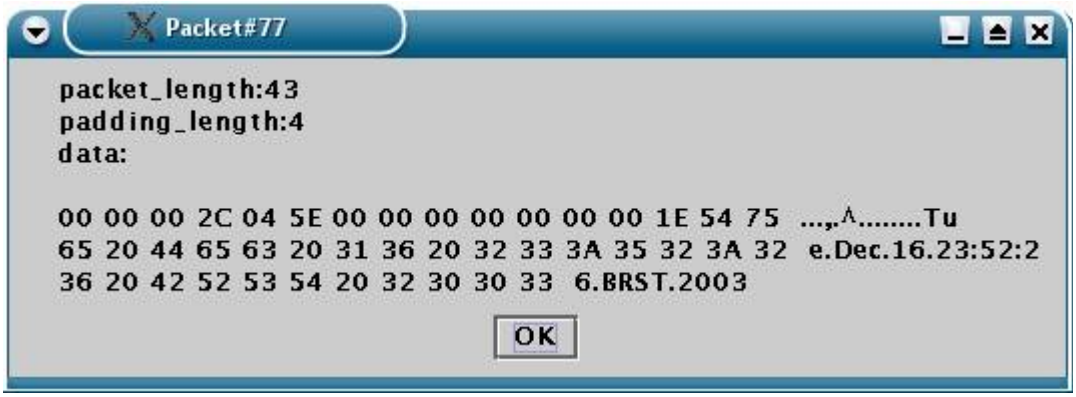
Alguns exemplos de pacotes (decriptados):



pacote SSH_MSG_USERAUTH_REQUEST para autenticação



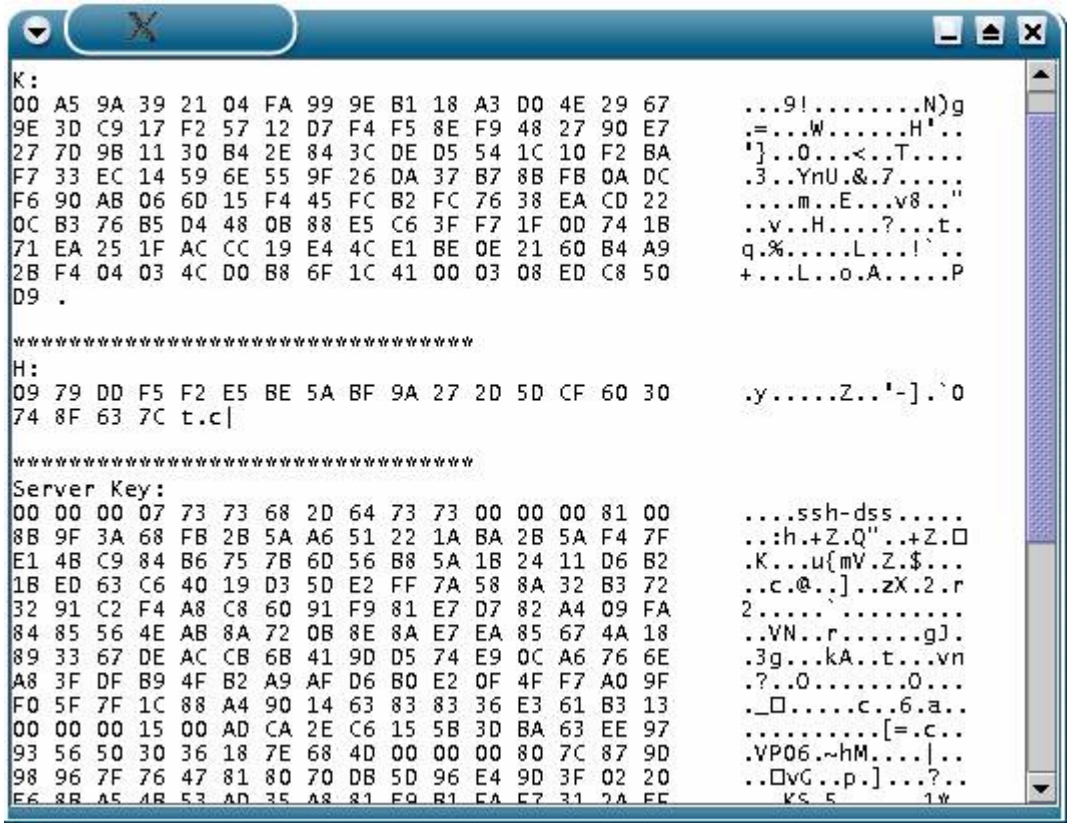
pacote SSH_MSG_CHANNEL_DATA enviando o commando “date” para o shell



resultado de “date” enviado para o shell:”Tue Dec 16 23:52:26 BRST 2003”

Na barra de status são mostrados os algoritmos utilizados para encriptar e para gerar os MAC’s. Nessa barra existe um botão (“KEX Status”) que mostra um janela com os dados da sessão (H, K, Pub.Key do Server). O MD5 fingerprint da chave pública é mostrada diretamente na barra de status.

barra de status com os dados da conexão



janela informando K, H e a Chave Pública do servidor

1.18 Recursos

O SSH necessita 6 MB de RAM mais o kernel do linux. Este limite deve ser redimensionado para servidores de acesso dedicao, 64MB de RAM deve ser suficiente para algumas centenas de conexões simultâneas.

O protocolo é aberto, o que dá a liberdade de implementações específicas dele de proverem novos recursos, através da utilização de outros protocolos paralelamente.

1.19 Prós e Contras

Prós	Contras
Protocolo aberto	Não recomendao para PCs lentos
Excelente velocidade do console	Necessidade de se digitar a senha toda vez que uma conexão com um servidor remoto for solicitada
Multi-Plataforma	
Conexão de dados criptografada entre cliente/servidor	

Cópia de arquivos usando conexão criptografada	
Suporte FTP criptografado.	
Suporte a compactação de dados entre cliente/servidor	
Controle de acesso das interfaces servidas pelo servidor SSH	
Suporte a controle de acesso TCP wrappers	
Autenticação usando um par de chaves pública/privada RSA ou DSA	
Algoritmo de criptografia livre de patentes	
Suporte a PAM	
Suporte as caracteres ANSI	

2. NX

2.1 Nome

NX

2.2 Descrição

Servidor de Acesso à Desktops Remotos usando novas sessões do X.

2.3 Autor/Empresa

O NX é desenvolvido pela empresa italiana NoMachine, sendo originalmente um produto pago (o servidor era licenciado e o cliente ficava disponível para download gratuito) tornando-se posteriormente completamente gratuito (tanto servidor como cliente gratuítos).

2.4 Site

http://www.nomachine.com

2.5 Função Resumida

O NX é um software para acesso à Desktop Remotos. Basicamente ele permite que servidores Linux e Solaris possam ser acessados remotamente e de modo eficiente por clientes que podem ser PCs, Thin Clients, Pocket PCs, entre outros. Cada um desses dispositivos cliente podem estar rodando Sessões do Windows, Linux e Solaris. O software provém o acesso remoto à servidores através de da inicialização de sessões do X remotas, onde são transmitidas as instruções e os pixmaps usados para montar a tela que será exibida no cliente. Esses dados são compactados usando um eficiente algoritmo próprio e encriptados usando o SSH, provendo performance e segurança tanto em links lentos (sobretudo conexões via ADSL ou modem)

quanto em redes locais, onde banda não é problema.

2.6 Funcionamento

2.6.1 Arquitetura

A arquitetura do NX é distribuída, sendo composta de um suite de tecnologias Open Source, desenvolvidas para se obter Computação em Rede de uma forma tão abrangente como, por exemplo, os *browsers Web* conseguem ter hoje. Consiste de uma camada de software servidor, que habilita qualquer computador Unix a trabalhar como um servidor de terminais. Já os software cliente são disponíveis para um porção grande de plataformas e Sistemas Operacionais (Windows, Unix, Sun).

2.6.2 X Window System:

A NoMachine escolheu construir o alicerce da Arquitetura distribuída no seu software no bem conhecido e largamente usado X Window System (ou simplesmente X), o sistema gerenciamento de janelas que está por trás do conhecido “Graphical User Interfaces” do Linux e Unix. Consequentemente, temos que o uso dos protocolos de comunicação do X estão presentes em sua arquitetura.

2.6.3 X Protocol Compression

A empresa fabricante do NX desenvolveu um exclusivo protocolo de compressão, baseado no já conhecido protocolo de compressão Zlib, e um conjunto de agentes integrados que possibilitam rodar sessões completas de desktop remoto, até mesmo em formato de tela cheia, usando limitadíssimas larguras de banda, como ocorrem em conexões usando modem, por exemplo. O Mecanismo de compressão do NX opera em três níveis do protocolo X:

a) Comprime o tráfego da rede em vários sentidos, como por exemplo, através do uso de avançados métodos de caching, redução de perda e compressão de imagens.

b) Reduz o chamado network round-trip a praticamente zero, maximizando o throughput.

c) Adaptação da largura de banda em tempo real, de acordo com as condições da rede.

O NX provém um protocolo de compressão com taxas de compressão do protocolo X variando entre 10:1 até 100:1.

2.6.4 RDP e RFB Protocols

A acessibilidade do NX não é restrita somente à Desktops e Servidores Linux. O NX encapsula e traduz no protocolo X o RDP (Remote Desktop Protocol) usado pela arquitetura do Windows NT/2000 Terminal Server Edition e também traduz o RFB (Remote Frame Buffer), o protocolo usado pelo VNC. Entretanto o mecanismo de compressão do NX oferece melhores performances quando roda aplicações X nativas. RDP e RFB sessões podem ser comprimidos num fator de compressão que varia de 2:1 até 10:1.

b) Protocolos: Como dissemos, O NX Server usa protocolos em vários aspectos: Na criação da Sessão Remota, é iniciada remotamente uma nova sessão do X (usando, portanto, o protocolo do X). Além disso é possível garantir um bom nível de segurança na transmissão dos dados e isso é obtido configurando-se a ferramenta para que ela criptografe os dados antes de serem enviados, com isso sendo feito usando o protocolo SSH. Outro exemplo de uso de protocolos se dá também com as sessões de Windows e de VNC. As sessões Windows e VNC são feitas “tunelando-se” os protocolos RDP (Remote Desktop Protocol) e RFB (Remote FrameBuffer) nativos destas arquiteturas via o NX, aumentando-se muito sua eficiência. Além disso o software contém uma tecnologia própria de compactação de dados baseada na tecnologia Zlib, provendo grande na compactação dos dados o que permite uma melhor performance da aplicação (menor consumo de banda para envio dos dados).

3. Recursos

O NX provêm recursos que até então outros programas de acesso à Desktops Remotos (como o VNC, por exemplo) não provém, como a transmissão de som e de arquivos, bem como o uso de aplicativos multimídia, jogos e interatividade (chats, bate-papos). Além disso provém o acesso à browsers (Internet), bem a abertura de sessões remotas dentro de sessões remotas.

4. Prós e Contras

Prós: As soluções atuais de computação baseada em servidor e rede são essencialmente adaptações para sistemas que não foram projetados para trabalhar em rede. Um comparativo com soluções baseadas em tecnologias como VNC e Rdesktop (Citrix) revela que estes protocolos foram implementados como contornos a uma deficiência nativa no sistema operacional em se trabalhar com eficiência em ambientes de rede. No caso do NX, toda arquitetura foi desenhada tomando por base a computação em Rede e aproveitando a eficiente arquitetura do sistema Unix, reconhecidamente mais estáveis e com melhor perfomance. Com NX, os benefícios da computação baseada em rede (Criptografia, Mobilidade, Compressão de até 1000:1, Múltiplas Plataformas de Acesso) estão ao alcance de qualquer um. Essa eficiente performance é observada se analisarmos, por exemplo, questões relacionadas ao seu modo de compressão de dados.

Devido ao alto grau de compressão dos mesmos que se obtém usando o NX, observamos que até mesmo novas sessões que são abertas usando conexões com limitações de banda (discadas, por exemplo) se comportam com excelente performance de acesso, se comparado com outras tecnologias concorrentes como o VNC, por exemplo.

Ao contrário de tecnologias como o VNC, o NX não fica tirando screenshots da tela e enviando pela rede. Como o NX abre sessões remotas do X e mapeia todos os pixels da tela para então comprimir esses dados, temos que há uma redução no tráfego gerado no envio dos dados que serão usados para se construir a imagem do Desktop remoto no cliente. Tudo Isso permite que estações de trabalho consideradas obsoletas ou ultrapassadas possam acessar novos sistemas, empregando a tecnologia de servidor de terminais.

Outro fator observado é que grande parte das empresas enfrenta dificuldades altos custos associados à manutenção e suporte de suas estações de trabalho, isso sem contar o alto custo de licenciamento. Isso faz com que o NX se apresente como uma solução barata (pois é gratuita) e que apresenta resultados muito satisfatórios.

Contras: Por ser um software que originalmente era pago, existe muito pouca documentação à respeito dificultando o entendimento de seu funcionamento, bem como a compreensão de sua arquitetura e seus componentes.

3. X Window System

3.1 Descrição

O X é um sistema de janelas e um protocolo que provê acesso a interfaces gráficas, suportado pelo Unix e Linux.

3.2 Autor/Empresa

O X surgiu em 1984 no MIT, uma das implementações dele (X.org) que atualmente está na versão 7.1 é mantida pela função X.org, e está disponível como software livre.

3.3 Site

www.x.org

3.4 Função Resumida

O X possui um framework para ambiente gráfico, para desenhar e mover janelas na tela, interagindo com o mouse, teclados e outros periféricos de entrada.

O usuário não tem contato direto com o X, esse contato é feito

através de programas clientes que dependem dele, como o KDE e Gnome.

O X permite que janelas (programas) que rodem nele possam ser executados remotamente pela rede, por diferentes usuários e telas. Essa característica de cliente e servidor garante um leque de possibilidades de acesso remoto.

3.5 Arquitetura

O X usa o modelo cliente e servidor (Ilustração 1), aonde um servidor comunica com vários programas clientes. O servidor aceita pedidos para desenhar janelas gráficas e de entrada via teclado, mouse ou até mesmo touchscreen.

O X pode exibir janelas do sistema local ou de sistemas de X remotos, permitindo programas clientes remotos comuniquem com o servidor de janelas X local.

A comunicação entre eles é feita através da troca de pacotes via socket na rede (Ilustração 2). A conexão é estabelecida pelo cliente, que envia os primeiros pacotes. O servidor responde aos pedidos do cliente com pacotes, aceitando ou não a conexão do cliente, ou até mesmo pedindo confirmação de acesso (controle de acesso), assim, estabelecida a conexão, 4 tipos de pacotes são trocados via rede:

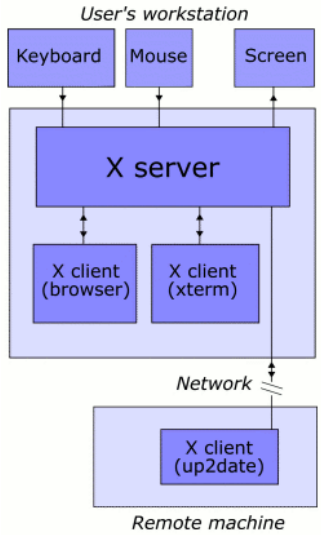


Ilustração 1: Arquitetura do X

1. Request (requisição): O cliente requisita informações do servidor
2. Reply (resposta): O servidor responde à requisição, mas nem todas requisições precisam ser respondidas.

3. Event (evento): O servidor envia um evento para o cliente, como a entrada de um teclado ou mouse, movimento de uma janela, etc.
4. Error (erro): O servidor envia um pacote de erro se a requisição for inválida. Como as requisições podem ser empilhadas, os erros podem ser gerados por requisições feitas anteriormentes.

Os pacotes de requisições e respostas podem ter um tamanho variado, mas os pacotes de evento e error tem tamanho fixo de 32 bytes.

O servidor X provê um leque de serviços básicos, para que o cliente possa realizar esses serviços e outros mais complexos, interagindo com o servidor.

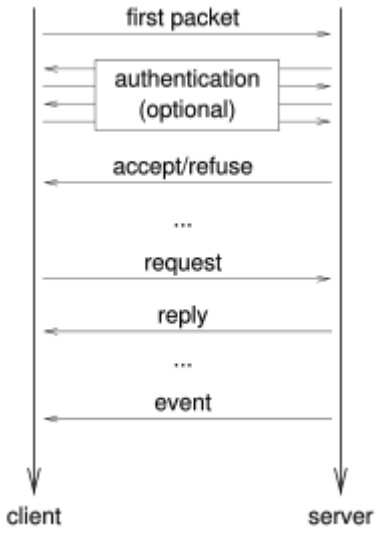


Ilustração 2: Troca de mensagens

3.5.1 Janelas

São interfaces gráficas que possui elementos como botões, menus, ícones, etc, componentes adequados a janelas.

Um janela só pode ser criada a partir de uma janela já existente (Ilustração 3), o que torna as janelas filhas de outras janelas, como um hierarquia. A janela principal é chamada de “janela root”, que é automaticamente criada pelo servidor, que nada mais é que a janela em tela cheia, atrás de todas as janelas.

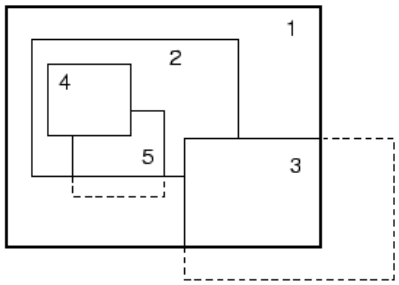


Ilustração 3: Arquitetura de janelas

3.5.2 Identificadores

Todos dados das janelas, incluindo fontes usadas são armazenados no servidor. Esses objetos são acessados pelo cliente através de um identificador reservado a cada objetivo, no qual o cliente já conhece.

Caso o cliente deseja que uma janela seja criada, ele faz uma requisição para o servidor para criar a janela passando um identificador. O servidor cria a janela e associa com esse identificador. O identificador pode ser mais usado mais tarde pelo cliente, para por exemplo, escrever uma frase nessa janela.

Identificadores devem ser únicos no servidor, independente do cliente, é um identificador único que não pode ser criado novamente por outro cliente.

3.5.3 Atributos e propriedades

Cada janela tem predefinido alguns atributos e propriedades, todos também armazenados pelo servidor e acessíveis pelo cliente através de requisições. Atributos são datas sobre as janelas, como tamanho, posição, cor de fundo e outros.

Propriedades são valores dos dados anexados as janelas, contrário aos atributos, propriedades não são pré-identificadas pelo protocolo do servidor. Podendo então, o cliente armazenar dados relevantes a ele nas propriedades das janelas.

Propriedades são caracterizadas por um nome, tipo e valor, são similares a variáveis, no qual uma aplicação cliente pode criar uma nova e armazenar valores. Propriedades são associados à janelas, duas propriedades com o mesmo nome podem existir em duas janelas diferentes, com o mesmo nome, valor e tipo.

Propriedades são bastante usadas pelos clientes e podem ser visualizadas com o comando xprop.

3.5.4 Eventos

Eventos são pacotes enviados pelo servidor para o cliente para comunicar que alguma coisa ocorreu. O cliente pode pedir para o servidor enviar os eventos para outro cliente, isso facilita comunicação entre clientes. Por exemplo: quando um cliente faz uma requisição para saber o texto que está selecionado no momento, um evento é enviado para o cliente que está manipulando a janela no momento.

Alguns tipos de eventos são sempre enviados para o cliente, mas a maioria dos tipos de eventos são enviados somente se o cliente deixar claro que necessita deles (exemplo: cliente quer ter informações do teclado, mas não do mouse).

3.5.5 Xlib

A maioria dos programas clientes comunicam com o servidor através da biblioteca Xlib, mas existem muitas outras bibliotecas que fazem isso (Motif, Gtk, Qt), algumas até usam a Xlib em conjunto para a comunicação.

3.5.6 Gerenciadores de janelas

Um gerenciador de janelas é um programa que controla a aparência das janelas / elementos gráficos. O papel principal de um gerenciador de janelas é prover uma interface amigável e padronizada com estilos para o usuário, contando também com aplicações úteis (exemplo: KUm gerenciador de janelas é um programa que controla a aparência das janelas / elementos gráficos. O papel principal de um gerenciador de janelas é prover uma interface amigável e padronizada com estilos para o usuário, contando também com aplicações úteis (exemplo: KDE, Gnome, WindowMaker).

3.6 Recursos

A comunicação do cliente com o servidor funciona de maneira transparente na rede, o cliente e o servidor podem funcionar na mesma máquina ou em máquinas diferentes, possibilitando uma interação de diferentes arquiteturas e sistemas operacionais, mas eles devem operar o mesmo sistema X.

A comunicação entre eles pode funcionar também de maneira segura, bastando criptografar a conexão através de um túnel criptografado (via OpenSSH).

Para iniciar um programa remoto e esse ser exibido (visto) localmente, basta o usuário mudar o seu “display”, apontando para um display remoto (servidor X).

Exemplo:
export DISPLAY=host:porta

Feito isso, basta o usuário invocar o programa, o cliente vai se conectar no servidor local da máquina e exibir a janela gráfica (Ilustração 4), aceitando também entrada de mouse/teclado.

O X permite que várias instancias funcionem com o servidor, permitindo que o usuário inicie sessões paralelas de ambientes gráficos.

É possível usar softwares de acesso remoto em conjunto com o próprio X, como por exemplo: VNC e NX.

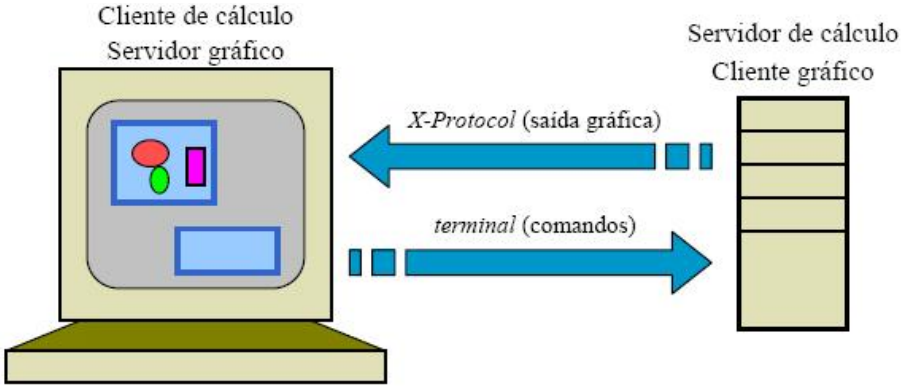


Ilustração 4: Exemplo de interação cliente e servidor

3.7 Prós e contras

Prós	Contras
Arquitetura escalável	Baixo desempenho para interações rápidas (jogos)
Comunicação via rede	
Suporte a extensões e módulos	
Drivers nativos	
22 anos de desenvolvimento	
Suporte a VNC, NX	

4. VNC

VNC - Computação em rede virtual (*Virtual Network Computing*).

4.1 Descrição

O desenvolvimento das redes de computadores se deu com o suceesso da idéia de disponibilizar acesso à recursos centralizados em um computador com alto poder de processamento, a partir de terminais simples de

baixo custo. A idéia dos criadores do VNC era levar este conceito além, e disponibilizar ao usuário não comente dados e aplicativos, mas um ambiente desktop completo que poderias ser acessado de qualquer computador conectado à Internet. Em contraste à onda de aplicações web, ondee o foco é acesso a recursos localizados em qualquer lugar do mundno através de um computador pessoal, o VNC permite que se acesse o computador pessoal de qualquer lugar do mundo.

4.2 Autor/Empresa

O VNC foi desenvolvido por membros do Olivetti & Oracle Research Labs (ORL) como uma feramenta interna para acesso aos computadores pessoais de qualquer infraestrutura computacional - desde o computador do escritório até terminais de acesso web, disponíveis em locais públicos.

4.3 Site

O site do RealVNC, produto atualmente mantido pelos autores do VNC original, é http://www.realvnc.com.

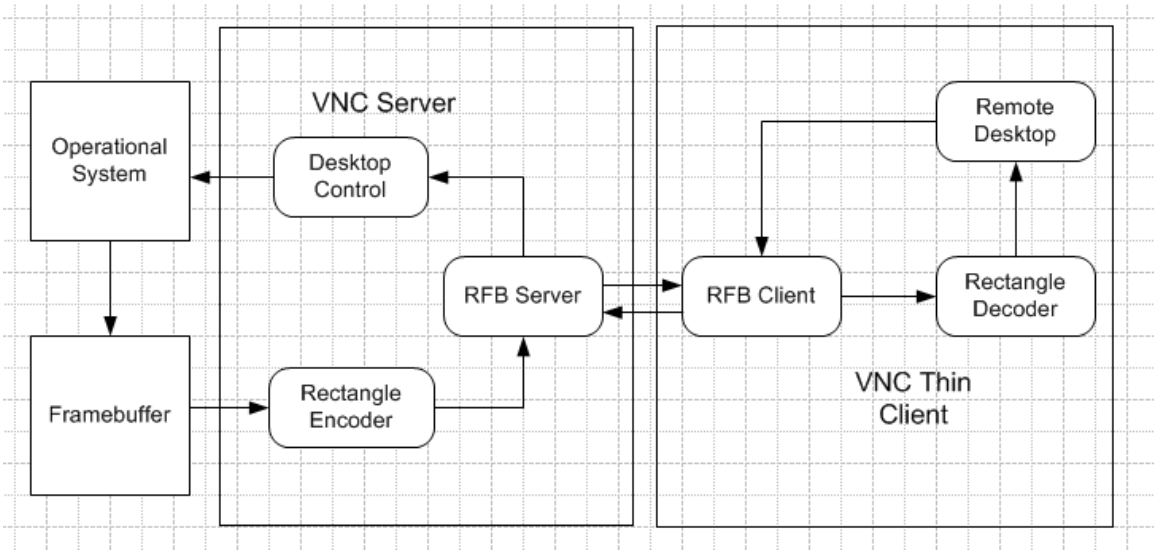
4.4 Função Resumida

O VNC é considerado um meio de computação móvel, onde não é necessário carregar nenhum dispositivo. Isto se dá devido a simplicidade do protocolo, que torna o lado cliente do sistema um *thin client*, aplicação pequena e simples que consome poucos recursos de processamento e memória. Isto abre possibilidades para implementação em plataformas *desktop*, dispositivos móveis e *applets* Java, disponibilizando acesso praticamente em qualquer lugar.

4.5 Arquitetura

Ilustração de um cliente VNC rodando em um ambiente móvel simulado:

Abaixo, uma ilustração da arquitetura do sistema VNC#, uma das implementações do protocolo RFB. Este sistema foi escolhido para ser exibido pois só implementa as funcionalidades básicas oferecidas pelo protocolo. Vale lembrar que este diagrama é específico da implementação, e pode variar de acordo com o projeto do *software*.



1. Framebuffer

O *framebuffer* é um dos módulos do sistema de vídeo dos computadores. Como o próprio nome diz, trata-se da memória onde são armazenados os quadros gerados pelo sistema de vídeo, antes de serem exibidos pelo

monitor de vídeo. Nos sistemas modernos, providos de funcionalidades gráficas 3D, os gráficos são originalmente gerados pelos *softwares* na forma de vetores. Estes sistemas possuem um *hardware* ou *software* que faz a conversão dos gráficos vetoriais para a forma matricial, gerando os quadros.

2. **RFB *Server***

O RFB *Server* é o módulo que implementa o protocolo RFB e é responsável pela troca de mensagens do servidor com o cliente. É uma aplicação que roda sobre um protocolo de transmissão confiável (tipo TCP).

3. **RectangleEncoder**

Este módulo faz a leitura dos dados do *framebuffer* e faz a codificação de acordo com um método especificado. Este método é negociado entre o cliente e o servidor no momento da conexão, e a escolha é baseada em variáveis como capacidade de processamento dos nós, variação do *frame* e atraso da rede.

4. **DesktopControl**

O módulo DesktopControl é o módulo que recebe as mensagens do cliente referente à entrada de usuário (eventos do mouse e teclado) e as repassa ao sistema operacional do servidor.

5. **RFB Client**

O RFB Client implementa o lado cliente do protocolo RFB, e é responsável por fazer a comunicação com módulo RFB Server. O lado cliente também é responsável por analisar o estado do meio de transmissão e ajustar as taxas de atualização e método de codificação conforme necessário.

6. **RectangleDecoder**

Funciona do modo oposto ao módulo RectangleEncoder, utilizando o método de codificação selecionado para decodificar os quadros recebidos do servidor e repassá-los à interface com o usuário.

7. **RemoteDesktop**

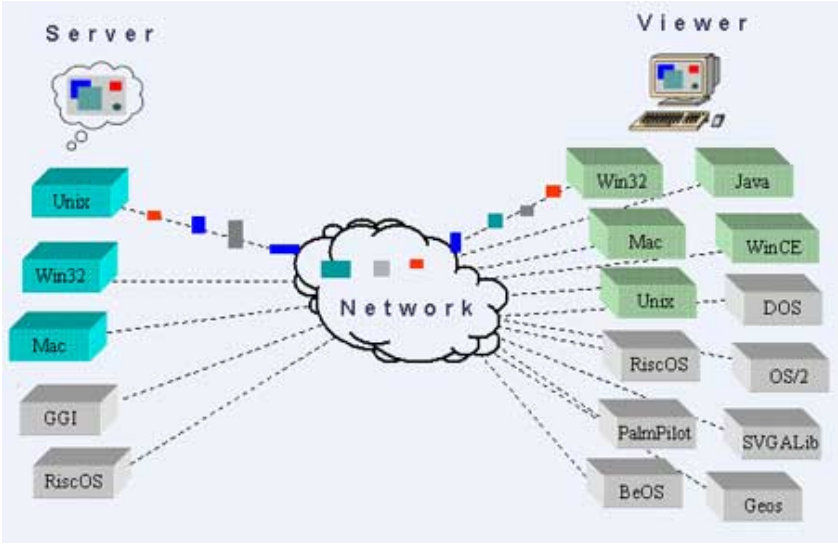
É a interface com o usuário propriamente dita. Este módulo exibe os quadros recebidos do servidor, e repassao ao RFB Client as entradas do usuário, que serão transmitidas ao servidor.

4.6 Protocolo

A tecnologia do sistema VNC é um protocolo simples, de acesso remoto à interfaces gráficas. O protocolo opera no nível do framebuffer, o que o torna independente de sistema operacional, plataformas gráficas e aplicativos,

proporcionando uma alta portabilidade. O protocolo opera sobre qualquer protocolo de transporte confiável, como o TCP/IP.

O sistema opera sobre uma arquitetura cliente-servidor, onde o lado servidor (VCN Server) fica responsável por enviar as mudanças no framebuffer da máquina onde está instalado ao lado cliente (VNC Viewer ou VNC Client).



O VNC é um sistema *thin client*, onde a união da simplicidade do cliente e do protocolo aberto torna possível implementá-lo sem dificuldades em qualquer plataforma.

Primitiva gráfica única

O lado cliente do protocolo é baseado em uma única primitiva gráfica:
Desenhe um retângulo de pixels em uma dada posição x,y.

A extrema simplicidade pode dar a impressão de que o desenho da interface é ineficiente. Mas este método permite a utilização de vários meios de codificação dos dados transmitidos, oferecendo uma grande flexibilidade em relação à variáveis do tipo: largura de banda da rede, capacidade de desenho da interface no cliente ou poder de processamento do servidor.

O método mais simples de codificação é denominado *raw encoding*, onde os dados são simplesmente enviados da forma como são gerados. Esta é a codificação mínima requerida pelo protocolo, que deve ser implementada pelos clientes e servidores, embora outras formas de codificação possam ser negociadas entre clientes e servidores no ato da conexão. Durante a etapa inicial da conexão (“handshaking”), os dois lados da aplicação se comunicam para definir a codificação de acordo com a capacidade das máquinas onde estão rodando e da largura de banda da conexão entre eles.

O problema é que estas não são as únicas variáveis que devem ser levadas em conta. A mais importante é a finalidade do usuário ao utilizar o

sistema. Por exemplo, a codificação *copy-rectangle* permite que uma vez que um retângulo da tela tenha sido exibido e ainda resida no framebuffer do cliente, ele pode ser exibido novamente sem que precise ser novamente transmitido pelo servidor (útil nos casos onde um único objeto se movimenta pela tela). Para a exibição de fotos, ou outras imagens de alta definição, a codificação dos frames em JPEG seria ideal. Ou para transmissão de vídeo ou outras situações que envolvam pixels em movimento, poderia-se utilizar codificação MPEG. Para exibição de texto, o cliente poderia salvar os caracteres exibidos para que possam ser reexibidos sem que sua imagem deva trafegar novamente pela rede.

Atualizações adaptativas

O mais próximo que se chegou de uma solução para o caso apresentado anteriormente (onde não é possível prever qual a codificação ideal quando não se sabe qual será a utilização) foram as atualizações adaptativas. Como o próprio nome diz, o sistema pode implementar um algoritmo que detecta padrões na movimentação dos pixels na tela, e se adapta alterando a codificação utilizada conforme o resultado da análise. O sistema pode trabalhar com múltiplas codificações simultaneamente, onde cada retângulo da tela pode ser transmitido codificado de forma diferente.

O sistema de atualizações adaptativas também reage a alterações na velocidade de transmissão, e reduz a freqüencia no envio de frames. Esta técnica implica em receber entradas do usuário, mas enviá-las somente quando requisitado pelo cliente. Assim, se o cliente detectar o baixo desempenho da conexão, e diminuir o número de requisições de atualização, todas as entradas de usuário que forem enviadas entre uma requisição de atualização e outra serão executadas, mas somente o resultado delas será exibido no cliente, sem que sejam mostrados os frames intermediários.

4.7 Recursos

O protocolo RFB permite somente a transmissão de quadros, sem possibilidades de transmissão de som ou arquivos, ou mesmo de uma forma segura de transmissão. Mas o protocolo aberto dá a liberdade de implementações específicas dele de proverem tais recursos, através da utilização de outros protocolos paralelamente. Um exemplo é o TIghtVNC, que permite a transmissão de som e arquivos. Atualmente, o RealVNC, implementado e mantido pelos pesquisadores da ORL (criadora do protocolo) implementa a transmissão de arquivos.

4.8 Prós e contras

Prós	Contras
Protocolo Aberto	Uso ineficiente de recursos de rede
Cliente pequeno que pode ser executado em	Protocolo limitado, que não permite extensão de

qualquer tipo de dispositivo computacional	funcionalidades (qualquer extensão necessita utilização de outros protocolos).
Implementações para todos os sistemas operacionais.	

5. Referências bibliográficas

RICHARDSON, T et al. Virtual Network Computing. IEEE Internet Computing, Cambridge, n. 3, p. 6-12, jan. 1998.

LUPTAK, M. Making VNC more secure using SSH. Virtual Network Computing, 1998. Disponível em: http://www.cl.cam.ac.uk/Research/DTG/attarchive/vnc/sshvnc.html. Acessado em: 06 out. 2006.

RICHARDSON, T The RFB Protocol. 3. ed. Cambridge, 2005, 43p.

SSH Home Page. Disponível em: http://www.ssh.fi/. Acessado em: 10 out. 2006.

Criando tuneis criptografados com SSH. Disponível em: http://www.dicas-l.com.br/dicas-l/20061017.php. Acessado em: 10 out. 2006.

Getting started with SSH. Disponível em: http://kimmo.suominen.com/docs/ssh/. Acessado em: 15 out. 2006.

Wikipédia. Disponível em: http://pt.wikipedia.org/wiki/SSH. Acessado em: 15 out. 2006.

O que é SSH. Disponível em: http://www.rnp.br/newsgen/9708/n3-3.html. Acessado em: 22 out. 2006.

Ylonen, T., "SSH Protocol Architecture", Disponível em: http://draft-ietf-architecture-15.txt, Acessado em: 30 out. 2006.

Ylonen, T., "SSH Connection Protocol", Disponível em: http://draft-ietf-connect-18.txt, Acessado em: 30 out. 2006.

Ylonen, T., "SSH Transport Layer Protocol", Disponível em: http://draft-ietf-transport-17.txt, Acessado em: 30 out. 2006.