ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

DESENVOLVIMENTO DE UMA APLICAÇÃO DE CHAT UTILIZANDO MULTICAST E CRIPTOGRAFIA SIMÉTRICA

Trabalho de Conclusão de Curso

Engenharia da Computação

Túlio de Lima Campos
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, maio de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

DESENVOLVIMENTO DE UMA APLICAÇÃO DE CHAT UTILIZANDO MULTICAST E CRIPTOGRAFIA SIMÉTRICA

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Túlio de Lima Campos
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, maio de 2005

Túlio de Lima Campos

DESENVOLVIMENTO DE UMA APLICAÇÃO DE CHAT UTILIZANDO MULTICAST E CRIPTOGRAFIA SIMÉTRICA

Resumo

A tecnologia Multicast aponta para uma grande utilização por instituições e empresas nos próximos anos, economizando largura de banda na Internet em aplicações de grupo como conferências de áudio e vídeo. Entretanto no seu modelo original, o Multicast apresenta problemas críticos de segurança. Este trabalho visa apresentar o funcionamento da tecnologia Multicast, bem como apresentar os problemas que enfrenta no enfoque de segurança. Além disso, mostra o desenvolvimento de uma aplicação de chat utilizando Multicast com criptografia simétrica para prover segurança na transferência de informações. É feito também um comparativo da aplicação desenvolvida com programas de chat tradicionais.

Abstract

The Multicast technology points to a great use by institutions and companies in the next years, saving bandwidth in group applications in the Internet such as audio and video conference. However, in its original model, the multicast technology presents security critical problems. This work aims at to present the functioning of multicast technology, as well as presenting the problems that it faces in the security approach. Moreover, it shows the development of a chat application using Multicast with symmetric cryptography in information transfers to improve security. A comparative between the developed application with common chat programs is also made.

Sumário

Índice de Figuras

Agradecimentos

Agradeço aos meus pais Everaldo e Walkíria, pelo carinho, amor, educação e credibilidade.

Aos meus amigos Igor e Luiz Antônio pelos anos de amizade e por terem compartilhado alegrias e desabafos ao longo desses últimos anos.

Ao meu orientador Adriano, pela paciência, sugestões, críticas e conselhos.

Aos colegas de curso pela troca de informações, por termos trabalhado sempre como uma equipe e por termos andado juntos esses anos na batalha pela formatura.

À Mariana por ter contribuído com a produção das ilustrações da monografia e as imagens da interface gráfica do Secure Multicast Chat e também por ter sido um dos maiores incentivos e estímulos para a realização deste trabalho.

À Zuleika por ter sido uma chefe extremamente compreensiva e pelas oportunidades que tem me oferecido.

Aos demais amigos que direta ou indiretamente me ajudaram.

Capítulo 1

Introdução

1.1 Considerações iniciais

Apesar de apontar para uma vasta utilidade e por economizar largura de banda quando usado de forma conveniente, o Multicast [35] em sua conjuntura atual é falho em vários aspectos de segurança [5, 8, 16, 17]. Não se pode controlar quem participa dos grupos, nem mesmo a quantidade de usuários no grupo. Além disso, a transmissão de dados é não-confiável, pois utiliza datagramas UDP, sendo suscetível a ataques de Denial of Service (DoS) especialmente quando aliado aos problemas anteriores. Para uma real popularização do serviço, soluções para estes problemas devem ser buscadas.

1.2 Motivação e objetivos

Atualmente, a maioria das empresas de grande, médio e até pequeno porte têm como parte de sua infra-estrutura redes locais de computadores conectadas à Internet. A maior demanda de serviços baseia-se em web e e-mail. Porém existe uma crescente demanda nos serviços multimídia, como teleconferência de áudio e vídeo, que requerem uma maior largura de banda e acesso de vários usuários a um mesmo tráfego de dados. Ao mesmo tempo estas empresas requerem sigilo na troca de informações com fornecedores e filiais que estejam participando das sessões e querem economia no acesso à Internet e infra-estrutura.

Com a finalidade de prover uma comunicação mais barata e conveniente em termos de largura de banda e infra-estrutura além do sigilo na troca de informações entre os participantes do grupo, temos como objetivo neste trabalho apresentar a tecnologia Multicast, sua forma de funcionamento na Internet atualmente e analisar as questões de segurança da tecnologia. Além disso, buscamos desenvolver uma aplicação de chat utilizando a tecnologia Multicast em conjunto com criptografia dos dados, para que as informações trocadas não sejam acessadas por indivíduos não autorizados. Esta aplicação pode ser utilizada como base ou módulo extra para outras aplicações, como videoconferência, utilizando a mesma tecnologia.

1.3 Organização da monografia

Esta monografia foi dividida em quatro capítulos. O presente capítulo introduz o tema proposto e relata as motivações e objetivos do trabalho.

O Capítulo 2 contém a fundamentação teórica do trabalho, onde são apresentados os principais aspectos do Multicast, seus protocolos e questões de segurança.

O Capítulo 3 trata do desenvolvimento da aplicação de chat Multicast utilizando criptografia, fala das tecnologias utilizadas para a implementação, apresenta as funcionalidades e problemas encontrados na aplicação e a seguir faz uma comparação com aplicações de chat comuns atualmente na Internet.

No Capítulo 4 é apresentada uma conclusão a respeito do trabalho desenvolvido e algumas sugestões para futuros trabalhos.

Capítulo 2

Fundamentação Teórica

2.1 Considerações iniciais

Neste capítulo será apresentada a fundamentação teórica na qual se baseia a elaboração desta monografia. Para desenvolver a aplicação proposta é essencial ter-se o conhecimento da tecnologia Multicast e seus aspectos relativos de segurança.

É interessante mostrar os aspectos da tecnologia Multicast e seu funcionamento na Internet, além dos problemas que enfrenta atualmente quando se trata de segurança. Desta forma, busca-se ter uma visão global da tecnologia e estímulo à produção de soluções para os problemas apresentados.

A questão de segurança em Multicast será abordada do ponto de vista das camadas de rede, transporte e aplicação.

2.2 Introdução ao Multicast

A maioria das aplicações comuns e mais antigas na Internet, a exemplo de e-mail e *browsers* utilizam a arquitetura cliente-servidor. Porém, em muitas das novas aplicações, como conferências de áudio e vídeo, trabalho cooperativo necessitam que um mesmo transmissor envie dados para vários receptores simultaneamente. Necessitou-se então de uma forma de comunicação em grupo que facilitasse o programador em relação a detalhes de processos individuais para cada membro do grupo. Surgiu então a proposta do modelo Multicast [22].

O conceito de endereçamento Multicast [25] para comunicação de grupo na Internet foi proposto por Steve Deering em sua tese de doutorado na Universidade de Stanford, e mais tarde desenvolvido no XeroxParc. Multicast para Internet foi adotado pela primeira vez no encontro da Internet Engineering Task Force (IETF), em Março de 1992.

O modelo IP Multicasting é uma extensão para o protocolo IP na camada de rede. A RFC 1112, Host Extensions for IP Multicasting [10], de autoria de Steve Deering 1989, descreve o IP Multicasting como: "a transmissão de um datagrama IP para um 'host-grupo', ou seja, um conjunto de zero ou mais hosts identificados por um IP de destino comum. Um datagrama multicast é enviado para todos os membros do grupo de destino utilizando o melhor esforço, como datagramas IP Unicast [21, 32]. A participação em um grupo é dinâmica, ou seja, os hosts

podem entrar e sair a qualquer momento. Não existe restrição de localização ou número de membros num grupo. Um host pode ser membro em mais de um grupo por vez". Adicionalmente, no nível de aplicação, um endereço de grupo pode ter vários *streams* de dados em diferentes números de portas, em diferentes *sockets* (mecanismos usados para a troca de dados entre processos, que podem estar todos em uma máquina local ou em diversas máquinas utilizando a rede), em uma ou mais aplicações. Múltiplas aplicações podem compartilhar um mesmo endereço de grupo em um host.

Embora se possa conseguir reproduzir o modelo de Multicast utilizando mensagens do tipo Unicast (um-para-um) ou Broadcast (difusão), a comunicação Multicast, em muitos casos, é preferível devido a alguns fatores, como:

- O tipo de endereçamento diminui o tráfego na rede, já que o emissor apenas manda uma mensagem para o grupo, em vez de gerar vários tráfegos Unicast idênticos para cada receptor. O pacote é replicado nos roteadores apenas quando necessário. Isso faz com que se diminua o consumo da largura de banda de transmissão da rede. A Figura 1 mostra a comparação entre os dois modelos ilustrando um host desejando enviar uma mesma informação para três receptores. No primeiro caso, três cópias do mesmo tráfego são geradas e então roteadas de acordo com os destinos. No segundo caso, apenas um tráfego é gerado e replicado quando necessário para chegar a todos os membros do grupo.

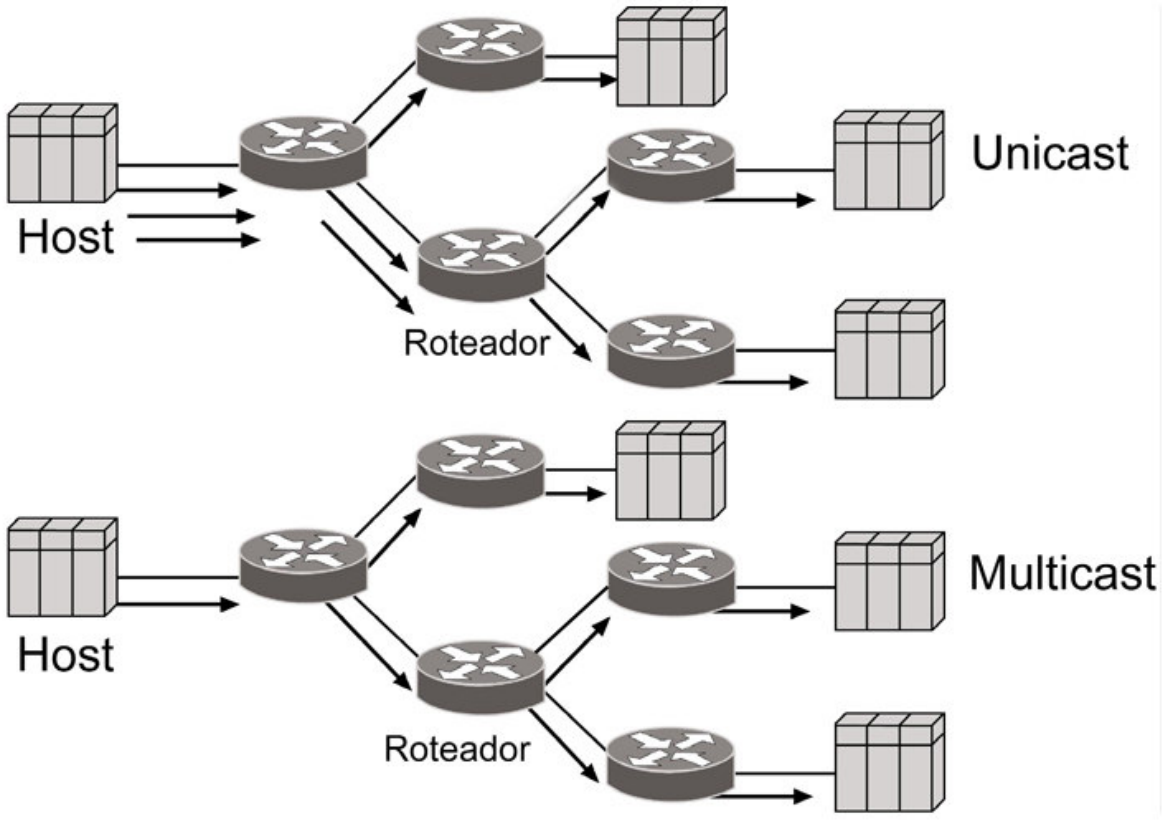

Figura 1. Tráfegos Unicast e Multicast para aplicações de grupo.

- O endereçamento IP utilizado para Multicast no Ipv4 é o de Classe D, que compreende os endereços 224.0.0.0 até 239.255.255.255. Ao contrário das outras três classes de endereçamento IP (A, B, e C), os 28 bits alocados para o ID de grupo multicast não têm nenhuma estrutura adicional. Um endereço do grupo do multicast é a combinação dos quatro bits mais significativos 1110 com um ID de grupo multicast. A Figura 2 mostra o

formato de um endereço de Classe D. Alguns endereços do grupo do multicast são atribuídos como endereços conhecidos pelo IANA (*Internet Assigned Numbers Authority*). Estes são chamados grupos permanentes. Isto é similar aos números de porta conhecidos do TCP e do UDP. Similarmente, estes endereços conhecidos do multicast são listados na *Assigned Numbers RFC*. Por o exemplo, 224.0.0.1 significa 'todos os sistemas nesta sub-rede', e o 224.0.0.2 significa 'todos os roteadores nesta sub-rede'. O endereço 224.0.1.1 do Multicast é para NTP, o Network Time Protocol, e assim por diante.

- Outra utilização é para a descoberta de recursos, ou seja, para saber se certos serviços estão ativos ou não na rede, enviando mensagens Multicast para vários potenciais hosts de destino que podem oferecer o serviço. O escopo dos pacotes pode ser limitado utilizando o campo *time-to-live* (TTL).
- Mais uma característica do Multicast é que ele oferece suporte a aplicações de Datacasting em transmissões multimídia, as quais estão se popularizando cada vez mais. Os sinais de áudio e vídeo são capturados, comprimidos e transmitidos ao grupo de hosts receptores, distribuindo-se os dados multimídia através de Multicast.
- No modelo Multicast pode-se integrar e sair dos grupos de forma flexível, tornando fácil a manipulação de integrantes.

Figura 2. Endereço Multicast.

Nas transmissões IP Multicast, os hosts emissores e receptores e a infra-estrutura de rede entre eles, devem dar suporte ao modelo, seguindo os seguintes requisitos:

- Oferece suporte à transmissão e recepção IP Multicast na pilha do protocolo TCP/IP;
- O software deve suportar IGMP (Internet Group Management Protocol) [7, 13] para comunicar requisições de entrada e saída de grupos Multicast;
- Placas e drivers de rede que devem filtrar em uma LAN os dados do endereço mapeados da camada de rede IP Multicast;
- Softwares de aplicação IP Multicast, como os de vídeo e áudio conferência.

Para a avaliação do tráfego IP Multicast em uma rede LAN, apenas as características acima são necessárias. Ao expandir o tráfego Multicast para uma rede WAN é necessário que todos os roteadores intermediários entre o emissor e o receptor ofereçam suporte ao IP Multicast. Além disso, os Firewalls precisam ser configurados para permitir este tipo de tráfego, porém isso não é algo difícil de ser feito. A Figura 3 mostra um diagrama do funcionamento de aplicações Multicast em uma rede onde a tecnologia está aplicada.

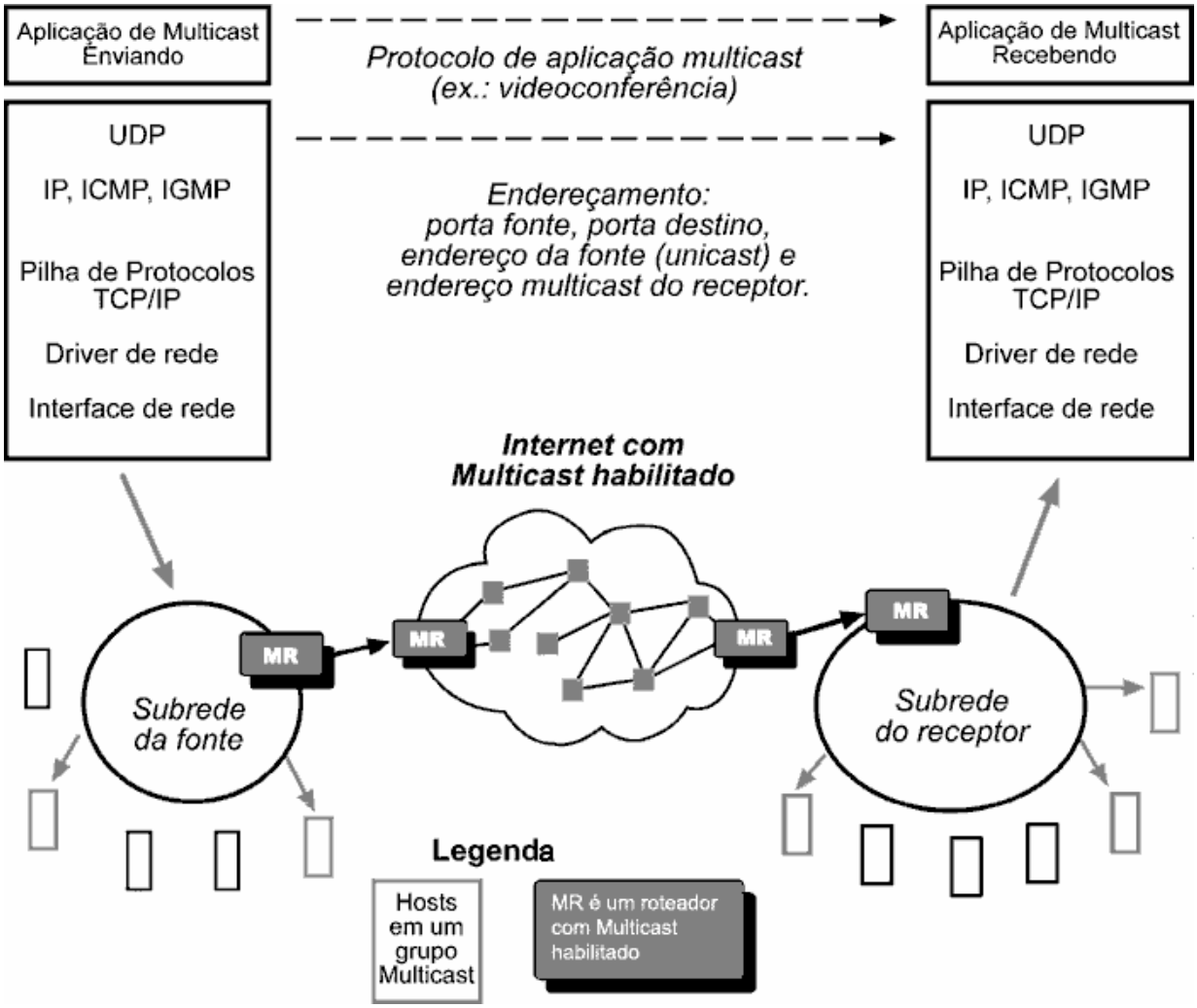

Figura 3. Aplicações numa rede Multicast.

A implementação de Multicast exige que funções básicas sejam definidas nos hosts e roteadores. Tais funções são incorporadas através do Internet Group Management Protocol (IGMP) que tem como finalidade fazer o gerenciamento dos grupos Multicast em conjunto com um protocolo de roteamento, como DVMRP [33], MOSPF [26], CBT [3, 4], PIM-SM [12, 14] e PIM-DM [1]. O funcionamento destes protocolos serão melhor detalhados no desenvolvimento deste trabalho.

2.3 MBone

O interesse em construir uma Internet com suporte a Multicast motivada pelo trabalho de Deering começou a se tornar realidade o final dos anos 80. Este trabalho levou à criação do Multicast na Internet e à criação do Multicast Backbone (MBone). Em março de 1992, o MBone carregou seu primeiro evento mundial quando vinte sites receberam áudio de um encontro da IETF em San Diego. Enquanto a conferência de software por si mesma representava uma considerável realização, o que ocorreu de mais significante foi o primeiro uso da rede virtual Multicast. A função de roteamento Multicast foi provida pelas estações de trabalho rodando um processo *daemon* chamado *mrouted*, que recebe pacotes Multicast encapsulados em Unicast em uma interface de entrada e repassa os pacotes em um conjunto apropriado de interfaces de saída. A

conectividade entre essas máquinas era provida usando túneis ponto-a-ponto (utilizados para encapsular mensagens de um protocolo em outro, usando as facilidades do segundo protocolo para atravessar parte de uma rede e extraindo a mensagem original na extremidade de saída do túnel, reinjectando-a na rede). Cada túnel conectava dois hosts configurados com *mrouted* por um único link lógico, mas podendo cruzar vários roteadores pela Internet. Quando se recebia um pacote, podia-se fazer broadcast para os membros locais ou enviar para outros hosts através dos túneis. As decisões de roteamento eram feitas usando o DVMRP (Distance Vector Multicast Protocol), que será explicado neste trabalho.

Na fase inicial do MBone, todos os túneis eram terminados em estações de trabalho e a topologia funcionava de forma que vários túneis passavam por um mesmo caminho físico. O roteamento Multicast desta forma sobrecarregava a rede. As primeiras versões do *mrouted* não suportavam *pruning* (podas), porém esta função foi agregada alguns anos depois. As podas servem para que roteadores não repassem tráfego Multicast adiante se não existem hosts associados.

Em 1992, o MBone cresceu tremendamente e passou a ser não apenas uma rede virtual no topo Internet, mas está se tornando integrada à própria internet. Adicionalmente a simples túneis DVMRP entre estações, o MBone agora conta também com suporte Multicast nativo, ou seja, alguns modelos de roteadores já são capazes de manipular e rotear pacotes Multicast. Além disso, outros protocolos de roteamento foram desenvolvidos e serão explicados mais adiante.

2.4 Protocolos de gerenciamento de grupos

Na utilização de Multicast, os hosts têm a responsabilidade de informar aos roteadores sobre a sua participação em grupos Multicast, possibilitando-os assim, contactarem outros roteadores, passando informações sobre as associações existentes, de forma que facilite o estabelecimento das rotas.

O protocolo padrão para realizar esta tarefa é o IGMP. A versão básica do IGMP data de 1988 e é agora um padrão da Internet. A mesma é descrita na RFC 1112 e conceitualmente possui três fases:

- Escolha do *Designated Router* (DR);
- Associação de um host a um par (Grupo, Fonte);
- Fiscalização da permanência dos hosts ao par e posterior desassociação.

A primeira fase se caracteriza pela escolha do *Designated Router*, ao qual vai ser atribuída a responsabilidade de desempenhar as funções do protocolo IGMP do lado dos roteadores.

A escolha ocorre após a troca de mensagens “Hello” entre os roteadores vizinhos. Geralmente o emissor com maior endereço IP assume as funções de DR. Além disso, cada roteador conectado à rede de múltiplos acessos envia mensagens “Hello” periodicamente, para adaptar-se às trocas de estado dos roteadores.

Para determinar se algum host em uma subrede local pertence a um grupo multicast, o DR envia periodicamente para a sub-rede uma mensagem multicast “Host Membership Query” a todos os nós em sua LAN, pedindo que relatem a participação nos grupos em seus processos através da mensagem “Host Membership Report”. Esta consulta é emitida ao grupo de todos os hosts (endereço de rede 224.0.0.1) e um TTL (*time-to-live*, ou número máximo de saltos) com valor 1 é usado de modo que estas consultas não sejam propagadas fora da LAN. Um host por grupo emite de volta uma mensagem de "Host Membership Report", enviado ao endereço do grupo. Assim todos os membros do grupo receberão a mensagem e não precisarão enviar esta mesma mensagem (logo, somente um membro relata a participação no grupo). Para que isso

ocorra, evitando congestionamento de tráfego, o tempo para enviar uma mensagem "Host Membership Report" é randômico.

Após a troca das mensagens, é incrementado um ponteiro para o par (Grupo, Fonte) na base de dados de grupos local, permitindo com isso que a rede receba as informações Multicast destinadas ao par (Grupo, Fonte). A troca de mensagens inicial do IGMP é exibida na Figura 4.

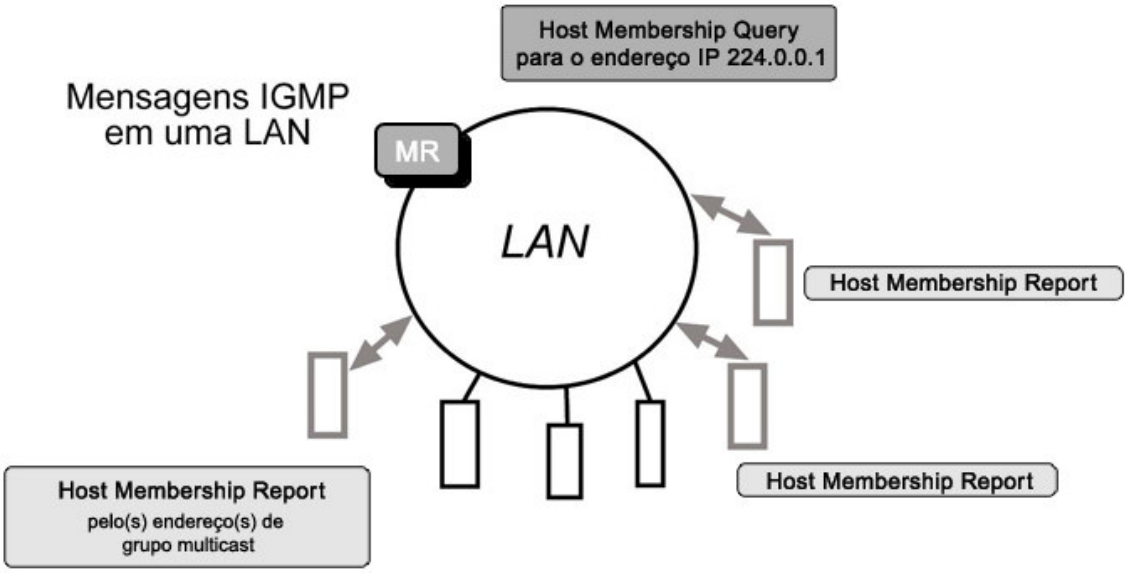

Figura 4. Troca de mensagens no protocolo IGMP.

A terceira fase é determinada pela sondagem da permanência dos hosts às associações estabelecidas e o processo de desassociação a um par (Grupo, Fonte). Isso ocorre de tempos em tempos, através da geração da mensagem “Host Membership Query”, por parte do DR, que pode ocasionar três comportamentos diferentes por parte dos hosts que estão associados à rede local:

- Uma nova associação, como se viu na segunda fase;
- Uma desassociação explicita através da mensagem “Leave Group” gerada por um host da rede local, ocasionando um decremento no ponteiro do par (Grupo, Fonte) dentro da base de dados de grupos local;
- Uma desassociação integral do par (Grupo, Fonte), indicado na mensagem “Host Membership Query”, caso não ocorra resposta por parte dos hosts da rede.

As mensagens do IGMP seguem um mesmo formato, em duas partes. Uma parte é para informações de controle e a outra é um endereço de classe D, como mostrado na Figura 5.

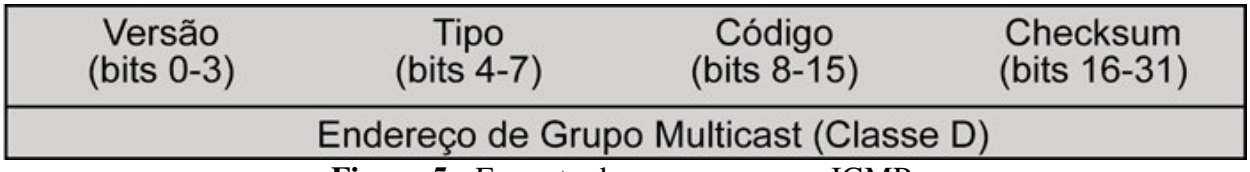

Figura 5. Formato das mensagens no IGMP.

2.5 Protocolos de roteamento intradomínio

O protocolo IGMP restringe-se ao diálogo local entre os hosts e o primeiro roteador Multicast alcançável. Para que o tráfego Multicast seja encaminhado por mais de um salto, o que é comum na Internet, a criação de uma árvore de distribuição é necessária, sendo responsabilidade do protocolo de roteamento.

Existem diferentes algoritmos para a criação da árvore de distribuição Multicast. No entanto, de uma maneira geral, os protocolos de roteamento Multicast mais difundidos podem construir dois tipos de árvore de distribuição: árvores por fonte (*source-based trees*, ou *source-specific trees*) ou árvores compartilhadas ou centradas (*center-based trees*). A diferença básica entre as duas é que, nas árvores de distribuição, para cada fonte de dados Multicast é necessária a criação e manutenção de uma árvore, enquanto que a árvore compartilhada é utilizada por várias fontes. Conseqüentemente, a raiz da árvore por fonte é o próprio nó fonte de tráfego, enquanto que a árvore compartilhada tem por raiz um nó arbitrário da rede (geralmente é estrategicamente localizado no centro da rede, num ponto onde se tenha maior largura de banda disponível).

Os protocolos de roteamento desenvolvem a função de atualizar as tabelas de rotas para a entrega de pacotes entre dois ou mais pontos, no momento em que ocorrem problemas em um enlace, ou um novo host deseja associar-se a um grupo Multicast. Os principais protocolos de roteamento IP Multicast intradomínio são o DVMRP, o CBT e os protocolos PIM Dense Mode e PIM Sparse Mode.

2.5.1 Distance Vector Multicast Protocol (DVMRP)

O protocolo de roteamento Multicast original cria árvores multicast usando uma técnica conhecida como inundação-e-poda [9]. Devido à forma como a árvore é construída ela é chamada de *reverse shortest path tree*. Os passos para a criação de uma árvore desse tipo são:

- A fonte envia em broadcast cada pacote em sua rede local. O roteador de borda recebe o pacote e envia por todas as interfaces de saída;
- A seguir cada roteador que recebe o pacote faz uma checagem de *reverse path forwarding* (RPF), ou seja, cada roteador checa se a interface de entrada em que o pacote multicast foi recebido é a interface que o roteador usaria como saída para atingir a fonte. Desta forma um roteador escolhe receber pacotes na interface que este acredita ter o caminho mais eficiente de volta pra a fonte. Todos os pacotes recebidos na interface em questão são repassados para todas as interfaces de saída e os outros são descartados.
- Eventualmente um pacote irá atingir um roteador conectado a alguns hosts. Este roteador, chamado de folha checará se existem membros do grupo em qualquer das sub-redes ao qual ele está conectado. O roteador descobre a existência de membros do grupo através de consultas do Internet Group Management Protocol (IGMP). Se não existem membros, o roteador folha envia uma mensagem de poda para a fonte na interface RPF, ou seja, a interface que o roteador folha usa para repassar pacotes para a fonte.
- Pacotes de poda são repassados de volta para a fonte, e roteadores no caminho criam um estado de poda para a interface na qual a mensagem de poda foi recebida. Se mensagens de poda são recebidas em todas as interfaces exceto a interface RPF, o roteador enviará uma mensagem de poda dele mesmo para a fonte.

Desta forma, árvores de caminho mais curto reverso são criadas. Protocolos inundação-e-poda são também conhecidos como protocolos de modo denso, pois são designados para funcionar melhor quando a topologia é densamente populosa com membros de grupo. Roteadores assumem que existem membros de grupo adiante e repassam os pacotes. Apenas quando

mensagens de poda explícitas são recebidas o roteador não mais repassa tráfego multicast. Se um grupo é densamente populoso, roteadores muito dificilmente necessitarão podar. A desvantagem principal dos protocolos de modo denso é que a informação de estado deve ser mantida para cada fonte em todos os roteadores na rede, sem considerar se membros de grupo receptores existem. Se um grupo não é densamente populoso, muitos estados devem ser armazenados na rede, e uma quantidade de banda considerável pode ser gasta.

2.5.2 Multicast Extensions Open Shortest Path First (MOSPF)

Roteadores que implementam MOSPF possuem uma imagem da topologia da rede, de acordo com o protocolo *Open Shortest Path First* (OSPF – Unicast), especialmente construído para distribuir informações da topologia entre roteadores de um mesmo sistema autônomo. O OSPF utiliza mensagens de estado do enlace (Link State Advertisement - LSA) que são trocadas por todos os roteadores da rede, de forma que cada um é capaz de construir um mapa atualizado da topologia da rede. Com esta informação, cada roteador é capaz de calcular o caminho mais curto entre dois nós quaisquer da rede, utilizando o algoritmo de Dijkstra. Os roteadores MOSPF mantêm uma imagem da topologia atual da rede através do protocolo OSPF. O MOSPF é construído sobre o OSPF versão 2, e mantém a compatibilidade com este. Utilizam-se do IGMP para monitorar as redes ligadas diretamente a eles, permitindo assim, que sejam criadas as tabelas de roteamento multicast, onde são mantidos os membros de um determinado grupo e especificado o roteador local que será responsável pela entrega dos pacotes Multicast a esses membros.

O MOSPF estende as mensagens de atualização do estado do enlace do OSPF com a informação de grupos Multicast ativos. Para que os roteadores possam manter as tabelas de roteamento é escolhido um roteador MOSPF, chamado Designated Router (DR), que envia mensagens IGMP à rede, com mensagens de consulta para obter informação de existência de membros de um determinado par (Grupo, Fonte) e espera por mensagens IGMP de resposta, as quais utiliza para atualização das tabelas de roteamento.

O DR é responsável ainda, pela transferência via flooding das informações existentes nas tabelas aos outros roteadores no domínio OSPF, assegurando com isso, que todos os pacotes originados remotamente possam ser transmitidos a outros membros de um determinado par (Grupo, Fonte). Isso é possível pela criação da árvore de caminho mais curto entre uma fonte de um grupo e seus demais membros.

Cada árvore de caminho mais curto é construída sob demanda, quando o primeiro pacote atinge um membro do par (Grupo, Fonte) destino. Após a construção da árvore, inicia-se o processo de “poda” dos galhos que não são necessários para a transferência dos pacotes. Se algum roteador desejar receber os pacotes após ter sido podado, este deve notificar seus vizinhos através de um pedido de reenxerto. Da mesma forma que cada roteador OSPF constrói a topologia de rotas Unicast, cada roteador MOSPF pode construir sua árvore de caminho mais curto para cada par (Grupo, Fonte). A utilização das mensagens de estado de enlace dispensa a utilização da inundação periódica de dados feita pelo DVMRP, mas por outro lado inviabiliza a utilização do MOSPF em redes muito grandes.

2.5.3 Protocol Independent Multicast - Dense Mode (PIM-DM)

Protocol Independent Multicast (PIM) foi dividido em dois protocolos, uma versão chamada PIM-DM e uma de modo esparso chamada PIM-SM. PIM-DM é muito similar ao DVMRP. Existem apenas duas grandes diferenças. A primeira é que o PIM (denso ou esparso) utiliza a tabela de roteamento unicast para fazer a checagem RPF. Enquanto o mecanismo DVMRP mantém sua própria tabela de rotas, PIM utiliza qualquer tabela unicast disponível. O nome PIM

é derivado do fato de que a tabela Multicast pode ser construída utilizando qualquer algoritmo de roteamento Unicast. PIM requer simplesmente a tabela de roteamento Unicast para existir e é independente do algoritmo usado para construí-la. A segunda diferença entre PIM-DM e DVMRP é que o DVMRP procura evitar o envio de pacotes desnecessários para os vizinhos que irão então gerar pacotes de poda baseados na falha de uma checagem RPF. O conjunto de interfaces de saída construído por um roteador DVMRP incluirá apenas os roteadores receptores que usam o dado roteador para alcançar a fonte (checagem RPF bem sucedida). PIM-DM evita essa complexidade, mas em troca disso todos os pacotes são repassados para todas as interfaces de saída. Pacotes desnecessários são geralmente repassados para roteadores que devem então gerar mensagens de poda devido ao resultado falho da checagem RPF.

2.5.4 Core Based Trees (CBT)

A próxima evolução no roteamento intradomínio foi desenvolver protocolos que amenizassem as desvantagens dos protocolos de modo denso. Uma nova classe de protocolos de modo esparso foi criada. Em vez de otimizar apenas para o caso de um grupo ter muitos membros, protocolos de modo esparso são designados para trabalhar de forma mais eficiente quando existem poucos e dispersos membros de grupo. Em vez de tráfego broadcast e utilização de mensagens de poda, espera-se que os receptores enviem mensagens explícitas para entrar nos grupos. Estas mensagens são enviadas para um roteador central chamado de core (núcleo). Espera-se que as fontes enviem seu tráfego para o mesmo roteador. O uso de um núcleo é um “ponto de encontro” para fontes e receptores, facilitando a criação da árvore multicast. O protocolo Core Based Trees (CBT) foi o primeiro a utilizar árvores centradas.

Quando um receptor deseja conectar-se a um grupo Multicast, este envia uma mensagem de enxerto (*join*) na direção do core. Esta mensagem cria um estado em cada roteador atravessado e faz com que uma mensagem de reconhecimento seja enviada ao roteador anterior, construindo a árvore Multicast. Assim que a fonte envia dados para o grupo Multicast (na direção do núcleo), estes são distribuídos na árvore a partir do primeiro roteador a recebê-los. Este roteador reenvia os dados em todas as interfaces de rede pertencentes à árvore Multicast, exceto a interface pela qual os dados foram recebidos, de forma a evitar loops.

O protocolo CBT foi inicialmente discutido na comunidade de pesquisa e está sendo padronizado pelo IETF. CBT utiliza o paradigma básico do modo esparso para criar uma única árvore compartilhada utilizada por todas as fontes. A raiz da árvore é o roteador core. Todas as fontes enviam seus dados para o núcleo, e todos os receptores enviam mensagens explícitas de participação nos grupos para o núcleo. Os roteadores na árvore de distribuição armazenam uma entrada por grupo, em vez de uma entrada por (fonte, grupo) em suas tabelas de roteamento. Desta forma, o CBT é mais escalável que o DVMRP e OSPF, em termos da quantidade de estados armazenada nos roteadores.

2.5.5 Protocol Independent Multicast - Sparse Mode (PIM-SM)

O PIM-SM é muito mais utilizado que o CBT. É similar ao PIM-DM pelo fato de que as decisões de roteamento são baseadas em qualquer tabela unicast que exista, mas a construção da árvore é um pouco diferente. O algoritmo de construção da árvore PIM-SM é realmente mais similar ao usado pelo CBT do que o usado no PIM-DM. Pode-se fazer a descrição da operação do protocolo de modo esparso da seguinte forma:

- Um núcleo chamado *rendezvous point* (RP, na terminologia PIM) deve se configurado. A decisão de quantos RPs deve-se ter e onde se deve coloca-los na rede é uma questão de planejamento de rede e está além do escopo deste trabalho. Grupos diferentes podem usar

diferentes roteadores como RPs, mas um grupo pode ter apenas um RP. A informação sobre que roteadores na rede são RPs e os mapeamentos dos grupos multicast para RPs devem ser descobertos por todos os roteadores. A descoberta do RP é feita utilizando o protocolo bootstrap. Entretanto, pelo fato do mecanismo de descoberta de RP não estar incluso na especcificação do PIM-SMv1, cada implementação de fabricantes de roteadores PIM-SMv1 tem seu próprio mecanismo de descoberta de RPs. Para o PIM-SMv2, o protocolo bootstrap está incluso na especificação do protocolo. A função básica do protocolo bootstrap, em adição à descoberta de RPs é prover robustez no caso da falha de um RP. O protocolo bootstrap inclui mecanismos para selecionar um RP alternativo se o RP primário estiver com problemas.

- Receptores enviam mensagens explicitas para o RP para entrar nos grupos. Uma única árvore compartilhada, com raiz no RP é formada para cada grupo. Como em outros protocolos multicast, a árvore é reverse shortest path tree – mensagens de entrada em grupos seguem um caminho reverso para o RP.
- Cada fonte envia pacotes de dados multicast encapsulados em pacotes unicast para o RP. Quando um RP recebe um dos três pacotes de registro, um número de ações é possível. Caso o RP tenha estado de repasse para o grupo, ou seja, existem receptores que entraram no grupo, o encapsulamento é retirado e enviado na árvore compartilhada. Entretando se o RP não tem estado de repasse para o grupo, ele envia uma mensagem register-stop para o RP. Isso evita o gasto de banda entre a fonte e o RP. Segundo, o RP pode querer enviar uma mensagem de entrada no grupo para a fonte. Estabilizando o estado de repasse entre a fonte e o RP, o RP pode receber o tráfego da fonte e evitar *overhead* de encapsulamento.

Estes passos descrevem o mecanimo básico usado pelos protocolos de modo esparso em geral e em particular do PIM-SM. Em resumo o fundamento principal do uso de um RP e o "ponto de encontro" para fontes e receptores. Receptores explicitamente entram na árvore compartilhada e as fontes se registram no RP.

Protocolos de modo esparso têm várias vantagens em relação aos de modo denso. Os de modo esparso tipicamente oferecem escalabilidade em termos de estados de roteamento. Apenas roteadores no caminho entre a fonte e o membro do grupo necessitam guardar estado. Protocolos de modo denso requerem estado em todos os roteadores na rede. Os protocolos de modo esparso são também mais eficientes, pois o uso explícito das mensagens de entrada nos grupos significa que o tráfego multicast flui apenas através de laços que foram explicitamente adicionados a arvore.

Porém, os protocolos de modo esparso têm algumas desvantagens. Estas estão mais relacionadas ao uso de RPs. O RP pode ser um ponto único de falha. Além disso, pode ser um gargalo de tráfego multicast. E tendo tráfego repassado de uma fonte para um RP e daí para os receptores pode significar que caminhos não muito eficientes podem existir na árvore multicast. O primeiro problema é quase totalmente resolvido com o protocolo de roteador bootstrap. O segundo e terceiro problemas são resolvidos no CBT utilizando árvores bidirecionais. PIM-SM resolve estes problemas provendo um mecanismo de não ser uma árvore compartilhada e sim uma *shortest path tree*. Esta mudança ocorre quando um roteador folha envia uma mensagem especial para a fonte. O *forwarding state* é alterado de forma que o tráfego flui diretamente para o receptor em vez de passar primeiro pelo RP. Esta ação ocorre quando o threshold da taxa de tráfego é violado.

Existem duas diferenças entre o CBT e o PIM-SM. A primeira é que o CBT usa apenas uma árvore compartilhada, e não foi projetado para usar shortest path trees. Segundo, o CBT utiliza árvores compartilhadas bidirecionais, enquanto PIM-SM utiliza árvores compartilhadas unidirecionais. Árvores compartilhadas bidirecionais envolvem uma maior complexidade, mas são mais eficientes quando pacotes que trafegam de uma fonte para o core atravessam desvios da

árvore multicast. Neste caso, em vez de enviar apenas até o core (para cima da árvore), pacotes podem também ser enviados para baixo. Mesmo o CBT tendo méritos técnicos significantes e estando tecnicamente empatado com o PIM-SM, poucos fabricantes de roteadores provêem suporte ao CBT.

Em resumo, os protocolos de roteamento Multicast utilizam ou um mecanismo inundação-e-poda ou um mecanismo explícito de participação nos grupos. Protocolos inundação-e-poda são mais conhecidos como protocolos de modo denso e usam uma shortest path tree com a raiz na fonte. Protocolos que usam mensagens de participação explícitas são chamados de protocolos de modo esparso e podem usar ou uma shortest path tree ou uma árvore compartilhada. Uma árvore compartilhada utiliza um ponto rendezvous para unir fontes e receptores.

A Tabela 1 mostra a relação dos protocolos apresentados anteriormente com o tipo de árvore de distribuição que utiliza e o mecanismo de participação nos grupos.

Tabela 1. Relação entre protocolos de roteamento Multicast, tipos de árvores de distribuição e mecanismos de participação em grupos.

Protocolo	Árvore de distribuição	Mecanismo de participação em grupos
DVMRP	por fonte	inundação-e-poda
MOSPF	por fonte	inundação-e-poda
PIM-DM	por fonte	inundação-e-poda
CBT	centrada	explícito
PIM-SM	centrada	explícito

2.6 Protocolos de roteamento interdomínio

O crescimento do MBone fez com que uma série de problemas aparecessem e se agravassem. A razão mais importante para isso é a crescente dificuldade de gerenciar a topologia virtual. Os mesmos problemas enfrentados pelo roteamento unicast se manifestaram no roteamento multicast. À medida que o MBone cresce, seu tamanho acaba se tornando um problema em termos de rotas e susceptibilidade a problemas de configuração. Como resultado, a comunidade multicast percebeu a necessidade de se implementar um roteamento hierárquico interdomínios. Em particular, o MBone se depara com problemas de escalabilidade e gerenciamento.

Em relação à escalabilidade, redes grandes são inerentemente instáveis. Mecanismos organizacionais pioram o problema por não prover agregação de rotas. Por essas duas razões, o MBone tem sofrido sérios problemas de escalabilidade. Em seu pico, o MBone chegou a ter mais de dez mil rotas e poucos hosts podiam ser representados em cada tabela de roteamento. E esses problemas não são novos. Como a Internet cresceu, o roteamento Unicast teve que ser profundamente alterado para continuar crescendo com estabilidade. As soluções de agregação de rotas e roteamento hierárquico foram bem sucedidas e seria interessante aplicar essas soluções também em Multicast.

O crescimento do MBone fez com que ele se tornasse mais difícil de gerenciar. O MBone não tem um gerenciamento centralizado. Pelo fato de o MBone ser uma rede virtual e que novos hosts podem conectar-se de qualquer lugar, dever-se-ia ter um procedimento formal para

adicionar novos hosts. Já que esse mecanismo não existe, o MBone cresceu randomicamente, com muitas ineficiências. Duas delas podem ser descritas a seguir:

- Gerenciamento da topologia virtual – O MBone é caracterizado como um conjunto de ilhas conectadas por túneis. A idéia sempre foi conectar essas ilhas da maneira mais eficiente, porém os túneis são frequentemente configurados de forma ineficiente. Este comportamento foi observado já no início do MBone, especialmente considerando o MCI Backbone. Para evitar o crescimento desses túneis ineficientes, engenheiros da MCI forçaram uma política na qual túneis através ou pra dentro da rede MCI deviam ser terminados em pontos de borda designados. O objetivo era limitar a quantidade túneis através de um link físico. O trabalho dos engenheiros da MCI foi um exemplo que ajudou a manter o MBone aceitavelmente eficiente por vários anos.
- Gerenciamento de política inter-domínio – Limites de domínio são outra fonte de problemas na topologia do MBone. O modelo na Internet atual é estabelecer limites de sistema autônomo (AS) entre domínios de Internet. ASs são geralmente gerenciados ou propriedades de organizações diferentes. Entidades em um AS são tipicamente não-confiáveis em outro AS. Como resultado, a troca de informações de rotas entre limites ASs são processadas com cautela. O relacionamento ponto-a-ponto entre ASs é feito utilizando o Border Gateway Protocol (BGP) [6], que provê abstração de rota e controle de política. Como resultado o uso em larga escala do BGP, existe um procedimento quando dois ASs desejam se comunicar. Pelo fato do MBone não prover um protocolo inter-domínio, ele não oferece proteção entre os limites de domínio. Quando existe uma única topologia usando túneis, problemas de roteamento podem ser facilmente espalhados através da topologia.

Da necessidade de se resolver estes dois problemas criaram-se as primeiras tentativas de se ter multicast interdomínio [20]. O multicast interdomínio tem envolvido a necessidade de se prover multicast de forma hierárquica, escalável e que trabalhe conjuntamente com todos os usuários de Internet. Protocolos que provêem a funcionalidade necessária têm sido desenvolvidos, mas a tecnologia é relativamente imatura. A solução imediata para roteamento multicast seria uma extensão do protocolo de roteamento unicast BGP em conjunto com protocolos adicionais necessários para construir e interconectar árvores através dos limites de domínio. Esta solução particular interdomínio em uso é considerada apenas imediata. É necessária uma solução definitiva em termos de escalabilidade. Trabalhos adicionais estão sendo feitos para a criação desta solução definitiva. Algumas propostas são baseadas no padrão IP multicast, enquanto outras tentam redefinir o serviço de forma a tornar o problema mais fácil de ser resolvido.

2.7 Segurança em Multicast

Os princípios básicos da segurança da informação [23, 24, 28, 31] são a autenticidade, confidencialidade, integridade e disponibilidade. A utilização destes princípios visa a redução de diversos tipos de problemas como uso indevido, fraudes, roubo de informações e outros, consequentemente aumentando a produtividade dos usuários através de um ambiente mais organizado em relação ao controle sobre os recursos de informática.

A autenticidade associa-se a identificação de forma correta de um usuário ou computador. O serviço de autenticação em um sistema deve assegurar ao receptor que a mensagem recebida é realmente procedente da origem informada no conteúdo. A implementação é feita através de senhas ou assinatura digital. A verificação de autenticidade é a medida de proteção de um serviço

ou informação contra a personificação por intrusos. Um ataque contra a autenticidade envolve alguma forma de personificação através de roubo e uso não autorizado de senhas.

Muitos tipos de sistemas não bloqueiam o acesso após várias tentativas mal sucedidas por erro de senha. Essa fraqueza em termos de segurança permite que um intruso tente várias vezes acessar o sistema com combinações de senha até que consiga uma que lhe permita entrar.

Confidencialidade significa proteger informações contra sua revelação para alguém não autorizado, seja interna ou externamente. Consiste em proteger a informação contra leitura ou cópia por alguém que não tenha sido explicitamente autorizado pelo proprietário daquela informação. A informação deve ser protegida qualquer que seja a mídia que a contenha, como por exemplo, mídia impressa ou mídia digital. Deve-se cuidar não apenas da proteção da informação como um todo, mas também de partes da informação que podem ser utilizadas para interferir sobre o todo. No caso da rede, isto significa que os dados, enquanto em trânsito, não serão vistos, alterados, ou extraídos da rede por pessoas não autorizadas ou capturados por dispositivos ilícitos.

O objetivo da confidencialidade é proteger informação privada (cidadãos, indústrias, governo, militar). Na comunicação, ela é obtida evitando-se a escuta (meio físico, topologia), ou se isto não for possível, evitando-se a inteligibilidade dos dados durante o processo de transmissão (cifra).

Uma rede de meios físicos compartilhados é uma rede na qual os pacotes são transmitidos para várias partes da rede à medida que trafegam dos pontos de origem para os de destino. As redes de meios físicos compartilhados impõem um tipo especial de risco de segurança, pois os pacotes podem ser interceptados em qualquer ponto dessas redes. A captura de pacotes dessa forma é conhecida como rastreamento da rede. Para o rastreamento de uma rede é preciso usar um dispositivo físico ou um programa. Os programas de captura de pacotes proporcionam uma interface com um dispositivo de hardware que é executado no modo promíscuo (*sniffer*), ou seja, copiando todos os pacotes que chegam até ele, independentemente do endereço de destino contido no pacote. Se um sniffer for instalado em alguma parte da rota entre dois hosts de uma rede, senhas e informações confidenciais podem ser capturadas, causando transtornos e prejuízos. Tal ação pode proporcionar, também, a ocorrência de futuros ataques contra autenticidade, usando senhas, nomes de usuário e endereços capturados por sniffers.

A integridade consiste em proteger a informação contra modificação sem a permissão explicita do proprietário daquela informação. A modificação inclui ações como escrita, alteração de conteúdo, remoção e criação de informações. Deve-se considerar a proteção da informação nas suas mais variadas formas, como por exemplo, armazenada em discos ou fitas de backup. Integridade significa garantir que se o dado está lá, então não foi corrompido, encontra-se íntegro. Isto significa que aos dados originais nada foi acrescentado, retirado ou modificado.
A integridade é assegurada evitando-se alteração não detectada de mensagens (ex. tráfego bancário) e o forjamento não detectado de mensagem (aliado a violação de autenticidade).

Ter as informações acessíveis e prontas para uso representa um objetivo crítico para muitas empresas. No entanto, existem ataques de negação de serviços, onde o acesso a um sistema ou aplicação é interrompido ou impedido, deixando de estar disponível, ou sendo o tempo de execução crítico, as operações são atrasadas ou abortadas. Disponibilidade consiste na proteção dos serviços prestados pelo sistema de forma que eles não sejam degradados ou se tornem indisponíveis sem autorização, assegurando ao usuário o acesso aos dados sempre que deles precisar. Isto pode ser chamado também de continuidade dos serviços.

Um usuário autorizado pode necessitar de um sistema que encontra-se indisponível, resultando em perdas tão graves quanto as causadas pela remoção das informações daquele sistema. Atacar a disponibilidade significa realizar ações que visem a negação do acesso a um

serviço ou informação, como por exemplo: bloqueio do canal de comunicação ou do acesso a servidores de dados.

Algumas questões em Multicast merecem especial atenção quando se trata de segurança. O fato de não se ter um controle da participação de hosts no grupo, nem da quantidade de hosts em um grupo [2] gera problemas graves quando se que trafegar dados confidenciais. Além disso, em um grupo não se sabe as características físicas dos membros, o que seria um problema se, por exemplo, está sendo transmitido um vídeo que consome muita banda e exige processamento alto e um host com equipamento e conexão deficiente entra no grupo para receber esse tráfego. Isso faz com que haja a necessidade de se ter controle de congestionamento para esses nós. Outro fator relevante é o uso de UDP para o tráfego Multicast, que não garante que os pacotes cheguem aos participantes do grupo. É notório que a implementação do projeto Multicast não teve preocupação alguma com a segurança do tráfego, a princípio.

2.7.1 Controle de dados e política de segurança

Vários fatores intrínsecos ao modelo IP multicast influenciam os modelos de segurança que devem ser usados. Podemos citar alguns como:

- Tipo de aplicação – as aplicações multicast geralmente são um-para-muitos ou muitos-para-muitos, além disso, diferentes tipos de dados podem ser trafegados. Como exemplos podem-se citar um sistema de transmissão de vídeo *pay-per-view*, onde a autenticação da fonte não tem muita importância e sim a confidencialidade do serviço para garantir que apenas os assinantes tenham acesso ao conteúdo da transmissão. Em videoconferência seria importante tanto autenticação quanto confidencialidade, assegurando a identidade do transmissor quanto a privacidade dos dados.
- Tamanho e dinâmica do Grupo – podem afetar a maneira como se aplica a segurança a um grupo de usuários. Em relação a dinâmica um grupo pode variar de poucos participantes a milhares em questão de segundos, seja por associações e desassociações explícitas ou por falhas em roteadores intermediários. Intuitivamente, a gerência de grupos menores e mais densos é mais fácil de gerenciar que grupos grandes. Para exemplificar podemos ter a exibição de um programa de TV e um vídeo sob demanda. Naturalmente o grupo de espectadores do programa pode ter várias associações ao início de um programa e várias desassociações ao final. Já o vídeo sob demanda tende a não apresentar um comportamento tão previsível.
- Escalabilidade – quanto mais escalável for o grupo e a aplicação, mais escalável deve ser o mecanismo de segurança utilizado. Isso significa que a segurança deve ser aplicada ao grupo e a todos os participantes sem deteriorar o serviço.
- Modelo de confiança – Caso seja utilizada criptografia, a questão de quem gera e gerencia as chaves deve ser discutida e bem planejada.

Com toda certeza estes fatores não esgotam a lista de possíveis aspectos relacionados com a segurança em multicast, mas servem ao menos de um guia inicial.

Assim como o tráfego Unicast, pacotes Multicast podem atravessar rotas públicas na Internet. Daí surge a necessidade de se fazer um controle dos dados enviados. Um método comum utilizado é a criptografia que pode ser simétrica ou assimétrica.

A criptografia simétrica é baseada em algoritmos que dependem de uma mesma chave, denominada chave secreta, que é usada tanto no processo de cifrar quanto no de decifrar o texto cifrado. Somente o emissor e o receptor devem conhecer a chave secreta, a qual necessita de proteção em relação ao ambiente externo, para que usuários não autorizados não tenham acesso a informação. Isso significa dizer que a segurança da comunicação depende da garantia de segredo da chave secreta, que só deve ser de conhecimento do emissor e do receptor.

A criptografia assimétrica baseia-se em algoritmos que utilizam duas chaves diferentes, relacionadas matematicamente através de um algoritmo, de forma que o texto cifrado pela chave 1 do par somente poderá ser decifrado pela chave 2 do mesmo par. As duas chaves envolvidas na criptografia assimétrica são denominadas chave pública e chave privada. A chave pública pode ser obtida pelo público em geral, enquanto que a chave privada somente deve ser de conhecimento de seu titular. Da mesma forma que no sistema de criptografia simétrica, a segurança da comunicação depende da garantia de segredo da chave privada, que só deve ser de conhecimento de seu titular. O emissor processa seu documento com a chave pública do receptor, que é conhecida. O texto cifrado somente poderá ser decifrado pelo receptor previsto, uma vez que somente ele tem a chave privada relacionada à chave pública que orientou a criptografia do documento emitido.

A idéia é baseada na cifragem da informação na fonte e decifragem nos receptores. Para que essa decifragem seja feita de forma correta, os receptores devem possuir as chaves que possibilitem a leitura correta das mensagens recebidas. Desta forma, dados capturados ao longo do caminho são inúteis para quem não conhece a chave de decifragem.

O grande problema da utilização de criptografia no controle de acesso surge quando este deve ser aplicado a dados transmitidos em alta velocidade. Numa transmissão de vídeo, por exemplo, a cifragem e decifragem de pacotes podem gerar atrasos no envio e exibição do mesmo por ser geralmente um processo mais lento.

Em IP Multicast é possível e útil separar autenticação e confidencialidade. Como já foi dito anteriormente, cada aplicação tem diferentes necessidades de segurança.

A autenticação pode ser dividida em autenticação por fonte e por grupo. A autenticação por fonte geralmente utiliza criptografia de chaves públicas (assimétrica), enquanto a autenticação por grupo, na maioria dos casos, utiliza criptografia de chaves privadas (simétrica). O uso de cada uma deve ser ponderado segundo seus desempenhos. Os algoritmos assimétricos em geral são mais lentos e seguros que os simétricos. Por outro lado a distribuição de chaves privadas em algoritmos simétricos é problemática, pois exige que a informação da chave de decifragem seja trafegada através de um canal extremamente seguro.

A depender da aplicação, o uso de chaves privadas também pode ser problemático já que qualquer host que participa do grupo pode se passar pela fonte, já que este possui a chave utilizada. Assim, é natural que para algumas aplicações sejam utilizados esquemas de chaves assimétricas, como assinatura digital. Essa solução, entretanto, apresenta overhead para assinar e verificar os pacotes e também de utilização de banda. Além disso, este problema pode ser agravado por ataques DoS (Denial of Service) que se dá quando um hacker inunda um receptor com pacotes que supostamente contém a assinatura. Já que a verificação exige um alto poder computacional, o receptor acaba não conseguindo mais sair do processo de verificação, tendo que analisar todos os pacotes maliciosos.

Em relação às políticas de segurança em Multicast, devem ser aplicadas as mesmas adotadas para tráfego Unicast. Porém existem questões diretamente utilizadas à problemática de operação Multicast, como as políticas de disseminação das chaves, controle de acesso e quais ações devem ser tomadas quando uma chave é comprometida.

Existem duas categorias abrangentes em relação à política de segurança multicast:

- Políticas com relação aos integrantes do grupo – envolvem as questões de quem pode participar ou não do grupo e em que condições, além de como os usuários serão autenticados e removidos. Aspectos como recursos computacionais mínimos também devem ser discutidos aqui.
- Políticas da garantia de segurança – tratam da disseminação inicial das chaves, alterações e procedimentos que devem ser tomados na ocorrência de erros.

É de extrema importância que a política como um todo seja coerente, livre de inconsistências e que possam cobrir todos os possíveis cenários encontrados pelo sistema.

2.7.2 Segurança na camada de rede

De forma genérica, a maioria dos problemas de infra-estrutura pode ser resolvida com base em soluções eficazes de autenticação e confidencialidade. A proteção da infra-estrutura multicast exige que sejam protegidos tanto a árvore de distribuição quanto o protocolo de transporte.

A árvore de distribuição está ligada diretamente ao roteamento. Está é responsável pela entrega de dados fim-a-fim, e a construção da mesmo está associada à escalabilidade dos grupos.

Dois tipos básicos de ameaças ao roteamento podem ser citados: ataques internos e ataques vindos da borda da rede.

Ataques oriundos da borda são originados em hosts conectados a roteadores-folha. Podem ser:

- Ataques vindos do emissor - a árvore de distribuição pode ser atacada por hosts enviando dados ilegítimos para o grupo. Como conseqüência, esses pacotes são entregues a todos os participantes do grupo, consumindo largura de banda. É interessante lembrar que não é necessário ser associado de um grupo multicast para poder enviar dados para o mesmo. Outra forma de atacar seria enviar dados numa taxa maior que a permitida, procurando inundar os integrantes do grupo.
- Ataques vindos do receptor – a idéia nesse caso seria de inchar a árvore de distribuição e consumir recursos dos roteadores (bandas dos links entre eles e informações de controle). O ataque se daria com muitos hosts não membros se associando ao grupo. O tráfego recebido, mesmo que cifrado é recebido e descartado pelos mesmos. Esse ataque pode ser considerado um DDOS (*Distributed Denial of Service*) por envolver vários hosts em colaboração.

Ataques internos são provenientes de dentro da árvore, tendo origem em roteadores invadidos ou invasores do grupo.

- Ataques com dados – assemelha-se aos ataques vindos do emissor, também enviando dados ilegítimos aos participantes do grupo. A diferença agora seria que o hacker é um membro do grupo.
- Ataques a dados de controle – neste caso, são injetados dados falsos de controle com o objetivo de atingir maliciosamente a árvore de distribuição. O hacker pode remontar a árvore de acordo com seus objetivos forçando a inclusão de um roteador próximo, para injetar tráfego, por exemplo.

Ataques ao roteamento dependem dos fatores que envolvem o roteamento e a forma como é construída a árvore de distribuição multicast. Logo, isto envolve:

- Protocolos de roteamento unicast e multicast utilizados;
- Utilização ou não da tabela de roteamento unicast por parte do protocolo de roteamento multicast, a exemplo do PIM;
- A topologia da rede;
- O tipo de aplicação multicast;
- O sentido do tráfego (uni ou bidirecional);

Propostas de proteção ao roteamento existem para o CBT e para o PIM. No caso do PIM-SM, já existe a solução de autenticação usando IPSec. A idéia seria que todos os roteadores PIM em um mesmo domínio utilizem a mesma chave simétrica. Algumas entidades, como o RP e o roteador bootstrap utilizam criptografia assimétrica. O roteador bootstrap pode, por exemplo, assinar digitalmente a lista de possíveis candidatos a RP.

2.7.3 Protocolos de Multicast Confiável

A distribuição de dados utilizando multicast em uma rede não confiável não garante a entrega confiável, que é exigência principal para diversas aplicações importantes, tais como, distribuição de software, informações financeiras, jornais eletrônicos e imagens médicas. O multicast confiável é também necessário em aplicações colaborativas. Conseqüentemente, o multicast confiável [15, 27, 30] é um problema importante que deve ser tratado de alguma forma.
Os protocolos de confiança do multicast não são novos na área dos sistemas de transmissão distribuídos e de satélite. Entretanto, a maioria destes protocolos aplica-se às redes locais e não se escalam bem em redes WAN, principalmente porque as entidades envolvidas no protocolo necessitam trocar diversas mensagens de controle com finalidades de coordenação. Adicionalmente não se tratam questões fundamentais como controle de fluxo, controle de congestionamento, latência fim-a-fim, e o atraso de propagação que tem um papel crítico em redes WAN.

2.8 Conclusão

Mostramos neste capítulo uma introdução ao Multicast e suas formas de endereçamento. Além disso, falamos do MBone que é a forma como está sendo utilizado o Multicast na Internet, utilizando o protocolo de grupo IGMP, alguns protocolos de roteamento intradomínio, dentre os quais o PIM-SM que é atualmente o mais utilizado para Multicast nativo. Falamos um pouco da idéia de protocolos interdomínio, os quais estão se desenvolvendo para fazer com que o Multicast funcione de forma eficaz e eficiente em toda a Internet. Ao final tratamos de algumas questões de segurança que envolvem Multicast e mostramos alguns aspectos problemáticos nessa tecnologia, já que foi projetada e implementada sem a preocupação com o impacto que essas falhas poderiam causar em certas aplicações.

Capítulo 3

Desenvolvimento do Chat Multicast com Criptografia Simétrica

3.1 Considerações iniciais

Esta parte do trabalho detalhará a implementação e testes de um software de chat utilizando a tecnologia Multicast e criptografia de chave privada baseada em senha para evitar que o texto claro trafegue através da internet. Desta forma, integrantes não autorizados do grupo e sniffers no caminho poderão acessar apenas a informação cifrada. Inicialmente será feita uma descrição de como o ambiente foi desenvolvido e quais conceitos e ferramentas foram utilizadas. O projeto chama-se Secure Multicast Chat, ou simplesmente SMC como poderá ser chamado a partir desta seção. Esta implementação interessa àqueles que desejam trafegar dados textuais utilizando Multicast com um mínimo de segurança, cabendo aos que pretendem utilizá-la analisar as vantagens e desvantagens fornecidas.

Ao final do texto é mostrado o ambiente desenvolvido em funcionamento e os resultados obtidos. Além disso, será feita uma comparação do SMC com programas de chat tradicionais em termos de custo, funcionalidade e segurança.

3.2 O Secure Multicast Chat (SMC)

O ambiente de chat desenvolvido SMC propõe-se a oferecer segurança no tráfego da informação, transparente ao usuário, através da utilização de Password-Based Encryption (PBE) com MD5 para hash [11] e utilizando o algoritmo DES na criptografia dos dados. O MD5 é um algoritmo de hash de 128 bits unidirecional desenvolvido pela RSA Data Security, Inc.

O algoritmo de hash é usado para gerar um valor do hash de alguns dados, como um e-mail, uma senha ou uma chave. A função do hash verifica qualquer modificação em um dado. O hash é um método para transformar dados de tal forma que o resultado seja exclusivo e não possa ser retornado ao formato original. Sua característica principal é a não-duplicidade de dados. Os mais usados algoritmos de hash são os MD2, MD4, MD5 e o SHA-1.

No ambiente SMC, o usuário pode associar-se a um grupo numa determinada porta associado a uma senha, a qual é utilizada para a cifragem e decifragem dos dados trafegados no grupo.

O SMC foi desenvolvido em Java [18, 19] utilizando as bibliotecas de redes java.net as quais implementam sockets Multicast e alguns pacotes do Java Cryptography Extension (JCE) javax.crypto para criptografia [34], as quais serão melhor detalhadas mais adiante, além de outras bibliotecas de suporte a métodos freqüentemente usados.

Alguns requisitos funcionais (RF) e não funcionais (RNF) do sistema proposto são:

- Prover um sistema de chat em Multicast que utilize criptografia para a troca de mensagens, de forma que usuários que também estejam associados ao mesmo grupo, na mesma porta só terão acesso ao texto claro se estiverem utilizando a mesma chave e o mesmo algoritmo para decifragem dos textos recebidos (RF);
- Fazer a cifragem e decifragem de forma transparente e automática para o usuário do chat, devendo os usuários apenas combinarem uma chave e entrarem com a mesma, para que a comunicação possa ser efetuada com sucesso (RF);
- Ser de fácil utilização. O usuário irá interagir com apenas uma tela que proporcionará todas as funcionalidades, desde a entrada e saída nos grupos à própria dinâmica do chat e visualização dos usuários participantes (RNF).

3.3 Sockets e Multicast em Java

Java é mais que uma linguagem de programação, é um ambiente completo de programação e execução, pois além de incluir a especificação da linguagem, possui uma API (Application Programming Interface) e uma máquina virtual. A API é formada por um conjunto de pacotes de classes que implementam funcionalidade não nativa da linguagem. A máquina virtual Java (denominada JVM, de Java Virtual Machine) executa um código Java através da interpretação de um arquivo em formato especial (extensão .class) contendo bytecodes. A JVM está presente, por exemplo, no comando java ou em navegadores Web.

Sockets são usados tipicamente para troca de dados via TCP ou UDP, o que é refletido no conjunto de classes que os representam. A API de sockets permite as seguintes operações: amarrar (bind) um socket a uma porta de comunicação TCP/IP; aceitar conexões de máquinas remotas na porta amarrada; esperar a chegada de dados; conectar a uma máquina remota; enviar e receber dados, e fechar uma conexão. Entretanto, em Java, a interface de Sockets é de mais alto nível, e esconde/agrega algumas dessas operações.

Em Java, a comunicação com sockets sobre UDP está baseada em duas classes principais: `DatagramPacket` permite criar um datagrama e "empacotar" dados no mesmo, assim como "desempacotar" os dados de um datagrama; `DatagramSocket` permite enviar e receber datagramas através da rede. Como não há distinção entre cliente e servidor, há apenas um tipo de socket (classe `DatagramSocket`); o construtor dessa classe cria um socket e faz a "amarração" do mesmo a uma dada porta, que pode ser especificada no construtor ou escolhida aleatoriamente pelo sistema.

A mesma classe `DatagramPacket` é usada para enviar e receber datagramas, variando apenas o construtor. Por exemplo, para preparar um datagrama a ser enviado, os argumentos são um array de bytes, o tamanho desse array, o endereço IP e o número da porta destinos; para receber, bastam os dois primeiros argumentos, sendo que o array estará vazio. Há outros métodos que podem ser utilizados. Por exemplo, `getData` e `setData` podem ser usados para obter ou configurar a carga do datagrama, respectivamente. Tipicamente, dados são "serializados" antes de

serem enviados no array de bytes do datagrama, de maneira a transformar tipos primitivos e objetos complexos em uma seqüência de bytes que possa ser recomposta no destinatário. Para tal, é possível utilizar as classes `ByteArrayInputStream` e `ByteArrayOutputStream`. Para enviar um datagrama a um grupo Multicast, utiliza-se um endereço IP da faixa reservada a Multicast. Java possui uma classe específica para Multicast, `MulticastSocket`, que estende `DatagramSocket`, e métodos que permitem entrar ou sair de um grupo, `join` e `leave`, respectivamente.

3.4 Criptografia Simétrica no SMC

Criptografia simétrica, também conhecida como criptografia de "chave secreta" é o tipo mais simples de cifragem. Uma única chave é usada, a qual deve ser mantida em sigilo, como o próprio nome diz. Para a cifragem a chave é utilizada para inicializar uma cifra. A cifra pode ser então utilizada para cifrar os dados passados pra ela. A decifragem é similar, a cifra é inicializada com a mesma chave, e os dados que passam são então decifrados.

A força da cifragem está no tamanho da chave. Para a criptografia simétrica o tamanho típico da chave está entre 40 e 128 bits, mas alguns algoritmos podem utilizar chaves maiores. O DES, que é provavelmente o algoritmo simétrico mais usado utiliza uma chave de 56 bits, o que não é suficiente para um sistema seguro em alguns casos. Chaves acima de 128 bits são mais recomendáveis, pois a quebra é bem mais difícil.

A criptografia simétrica pode ser utilizada em várias áreas. É muito mais rápida que a criptografia assimétrica, ou criptografia de chaves públicas (às vezes por um fator de 1000) e por esta razão é recomendada em situações onde muitos dados precisam ser transformados. Cifragem de arquivos, pacotes de rede e bancos de dados são aplicações plausíveis para criptografia simétrica. A grande fraqueza está na simetria da chave e na forma como esta chave vai ser comunicada às pessoas que realmente devem receber os dados. A mesma chave utilizada para cifragem é utilizada para decifragem de forma que se alguém consegue cifrar, consegue também decifrar e vice-versa. Ao ser enviada uma mensagem cifrada simetricamente, tanto o emissor quanto o receptor devem primeiramente entrar num acordo de uma chave. Para o SMC foi utilizada a criptografia simétrica, porém poderia ter sido usada a criptografia assimétrica sem degradação notável do desempenho do sistema já que os dados são apenas textuais, ou seja, manipula-se poucos dados. Para uma aplicação que utilize vídeo, não é recomendável o uso de criptografia assimétrica. A Figura 6 mostra como funciona o processo de cifragem e decifragem de dados na criptografia simétrica.

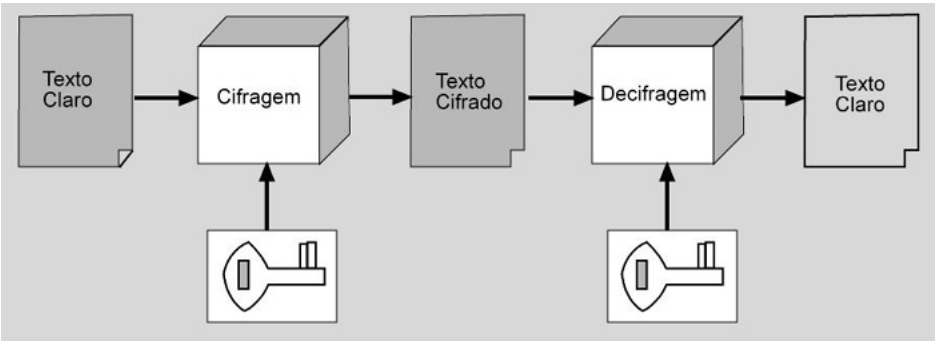

Figura 6. Processos de cifragem e decifragem.

3.4.1 Password-based Encryption em Java

Password-based Encryption (PBE) [29] usa uma única senha como a chave, ou seja, é uma forma de criptografia simétrica. Isto é conveniente porque a segurança depende apenas do usuário e não de um meio físico onde a senhas são armazenadas e há a possibilidade de invasão e roubo das mesmas. Infelizmente, password-based encryption no geral não é tão seguro quanto algoritmos com chaves binárias como TripleDES ou Blowfish. Para demonstrar isto, vamos dizer que usamos Blowfish com um tamanho de chave de 128 bits. Isso dá-nos um espaçamento de chave de 2^{128}. Password-based Encryption usa tipicamente caracteres do ASCII. A senha de um usuário tem, em média, seis caracteres de comprimento. Se forem utilizadas apenas letras minúsculas, há somente 26^6 chaves possíveis, ou aproximadamente 2^{28}. Adicionando caracteres maiúsculos e símbolos especiais, pode dificultar a quebra, mas dificilmente se aproxima do espaçamento de chave de um bom algoritmo simétrico.

Para adicionar à sua insegurança, a maioria das senhas que as pessoas utilizam são palavras cotidianas simples. Se um atacante tentasse quebrar uma senha, tentaria provavelmente cada palavra de um dicionário, que poderia ser gerado mais rapidamente em um computador. Isto é conhecido como ataque do dicionário e é incrivelmente bem sucedido. Existem ferramentas disponíveis que podem automatizar um ataque do dicionário - algumas inclusive testam combinações de palavras, diferindo na capitalização das letras, e do uso dos números e dos alguns símbolos. É interessante testar algumas destas ferramentas e ver o quão segura é uma senha, antes de utilizá-la. Uma senha boa deve ter ao menos oito caracteres de tamanho e usar capitalização, números, e símbolos, como "z1Hseg0+q". É importante que a senha seja simples o bastante para memorizar, entretanto nunca deve ser escrita em nenhum lugar.

O PBE utiliza uma combinação de hash e uma cifragem simétrica normal. É feito um hash da senha utilizando um message digest como MD5, então a saída deste message digest é usada para construir uma chave binária para um algoritmo, como o DES, como mostrado na Figura 7.

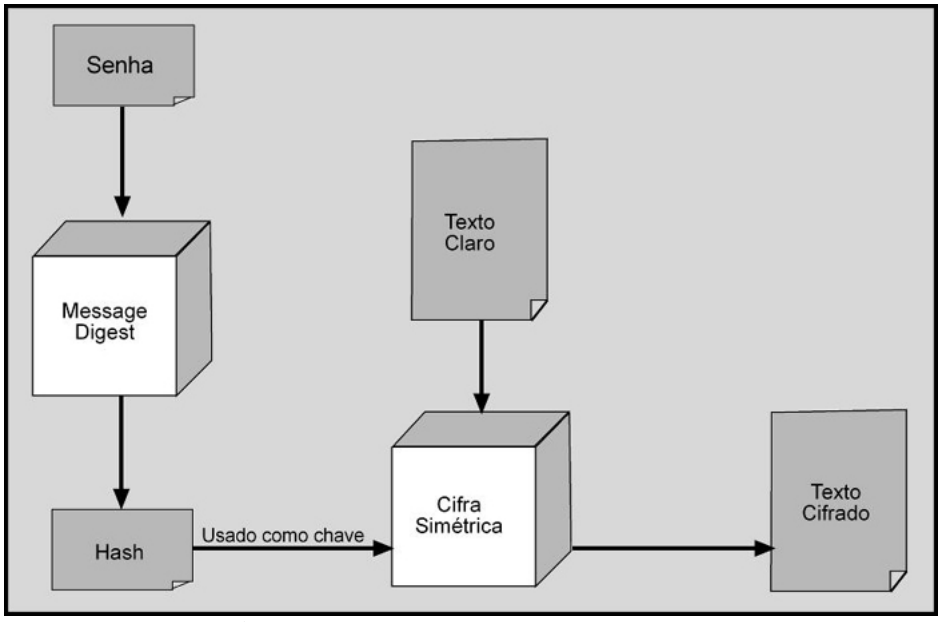

Figura 7. Password-based Encryption

Como mencionado, um dos problemas com PBE é que é possível criar uma lista de senhas pré-compiladas, fazer hashs e se ter um conjunto de chaves pronto pra atacar dados cifrados. Isto permite que um atacante possa testar normalmente várias chaves com muita rapidez e perceber se alguma delas pode decifrar os dados. Existem duas técnicas usadas no combate a este tipo de

ataque: salts e contadores de iteração. Um salt é um valor randômico anexado à senha antes que seja feito o hash para criar a chave, e o contador de iterações é o número de vezes que será feito hash com o salt e a senha. Examinaremos com mais detalhes como cada uma destas técnicas são usadas para prover uma maior segurança.

3.4.2 Salts e Contadores de Iteração

Adicionando dados randômicos à senha, antes de fazer o hash, aumenta grandemente o número de chaves possíveis criadas a partir de uma senha. Sem um salt, ao se fazer um hash, será transformada em uma outra específica. Adicionando oito bytes de dados randômicos antes de se fazer o hash na palavra, então esta palavra pode se tornar algo com 264 possibilidades. Isto significa que é efetivamente impossível de se criar um dicionário pré-compilado, já que o mesmo seria imenso. Em contrapartida, o atacante é forçado a fazer computação de hash em letras, números e símbolos de forma randômica, o que fará com que o mesmo precise de muito mais tempo para quebrar a senha.

O salt é gravado com os dados que são cifrados. A cada vez que novos dados são cifrados, um novo salt é gerado. Isso significa que a mesma frase será cifrada para um valor diferente a cada vez que esta passar pelo algoritmo, mesmo que a mesma senha seja utilizada. Para decifrar. O salt deve ser extraído dos dados cifrados e então combinados com a senha para criar a chave de decifragem.

Um exemplo do uso de salt é a checagem de senhas em sistemas Linux. Tipicamente utiliza-se MD5 para hash de senhas com um salt randômico. O salt é adicionado ao hash resultante. Este conjunto é então escrito no arquivo de senha. Os primeiros caracteres são o salt e os restantes é a senha em hash com o salt. Quando alguém tenta entrar no sistema, este lê a senha que é digitada e o registro no arquivo de senhas. Se o hash da senha digitada for igual aos caracteres finais do registro, a senha está correta. Um exemplo será mostrado mais adiante para uma melhor compreensão do uso de salt no PBE.

O contador de iteração é uma tentativa de aumentar o tempo que um atacante irá gastar para testar possíveis senhas. Se tivermos um contador de iteração de um mil, necessitaremos fazer o hash senha mil vezes na senha, o que é mais trabalhoso que fazer apenas uma vez. Neste caso um atacante precisará de mil vezes mais recursos para quebrar a cifragem. A escolha do número de iterações deve levar em consideração os requisitos de segurança e desempenho desejáveis à aplicação.

3.5 Utilização de PBE no SMC

Vamos mostrar como foi escrita a classe que utiliza PBE para cifragem e decifragem no projeto SMC. Foi utilizado um contador de iteração com valor de um mil e um salt randômico de oito bytes ou 64 bits. Quando escrevemos um texto, os primeiros 64 bits serão o salt que será necessário para os receptores decifrarem. Ao decifrar, utilizamos os primeiros 64 bits como o salt e decifraremos o texto que começa a partir do 64º bit. Note que o salt não é mantido em segredo já que é gerado randomicamente e cada bloco de texto a ser cifrado.

Já que as saídas devem ser exibidas utilizaremos a codificação de BASE 64, que transformam dados binários em caracteres ASCII.

3.5.1 Codificação de Base 64

Dados binários são tipicamente armazenados em bytes de oito bits. Caracteres ASCII são armazenados em sete bits, então se queremos exibir binários como ASCII, perderemos no mínimo um bit por byte. Codificação de BASE 64 é uma forma de resolver este problema. Bytes de oito bits são convertidos em blocos de seis bits e então para caracteres. Seis bits são utilizados de forma que alguns caracteres de controle podem ser utilizados para indicar o fim dos dados. Os caracteres codificados podem ser exibidos na tela e convertidos de volta para binário sem dificuldade. Evidentemente, já que estamos mudando de blocos de oito para blocos de seis bits, teremos mais blocos – três bytes transformam-se em quatro caracteres de vice-versa.

Existe um codificador/decodificador de BASE 64 no pacote sun.misc, o qual é utilizado neste projeto. Já que não está incluso em um pacote da API de Java, sua localização pode mudar em uma versão futura do Java.

O código apresenta cifragem e decifragem. Na cifragem será necessário uma senha e um texto claro e na decifragem, a senha e dados cifrados. Criaremos um salt para a cifragem e adicionaremos ao texto cifrado, como mostra a Figura 8.

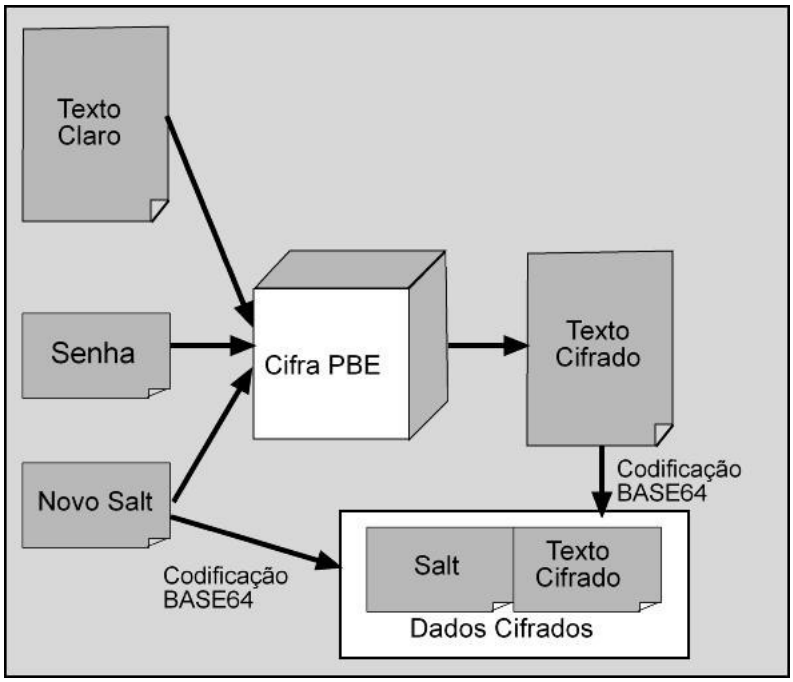

Figura 8. Cifragem com Salt e Contador de iteração

Ao decifrar, pegamos um bloco de dados cifrados e separamos o salt e o texto cifrado. Então podemos usar a senha e o salt para inicializar a cifra que pode decifrar a mensagem cifrada para o texto claro original como mostra a Figura 9.

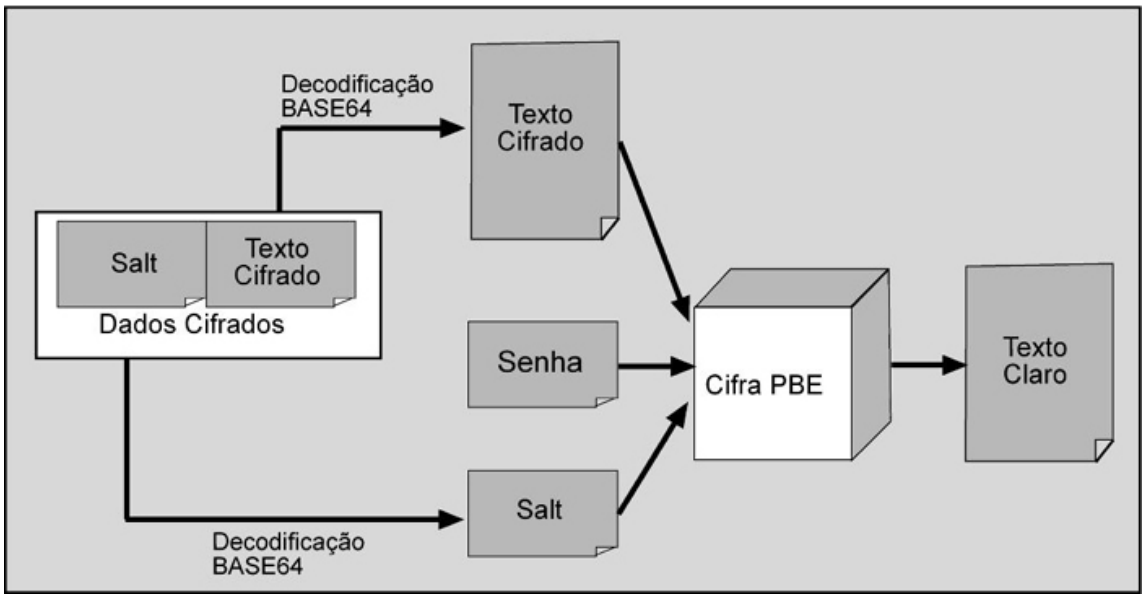

Figura 9. Decifragem com Salt e Contador de iteração

3.5.2 Cifragem e Decifragem dos dados

PBE em Java requer o uso de duas classes do pacote `javax.crypto.spec`, `PBEKeySpec` e `PBEParameterSpec`, assim como `javax.crypto.SecretKeyFactory`.

Utilizamos `PBEKeySpec` para criar uma chave baseada na senha, usando uma instância de `SecretKeyFactory`. Esta é uma das poucas classes no JCE que podemos realmente chamar o construtor padrão e pega um array de caracteres como argumento.

```
char[] password = "teste".toCharArray();
PBEKeySpec keySpec = new PBEKeySpec(password);
```

Para realmente usar o `PBEKeySpec` como chave, precisamos passá-lo por uma `SecretKeyFactory`, que irá gerar a chave, dada a especificação da chave, chamando o método `generateSecret`.

Criamos uma instância de `SecretKeyFactory` chamando o método `getInstance` com o nome do algoritmo que precisamos. Segue um trecho de código criando uma `SecretKey` a partir de `PBEKeySpec`:

```
SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance("PBEWithSHAAndTwofish-CBC");
SecretKey theKey = keyFactory.generateSecret(keySpec);
```

Para criar uma cifra que utilize PBE, precisamos passar o salt e o contador de iteração para a instância da cifra. `PBEParameterSPec` é um *wrapper* para o salt e o contador de iteração. É fornecido para a cifra na inicilização, juntamente com a chave. Assumindo que o salt e as iterações foram previamente definidos, mostramos como se obtém a cifra usando `PBEParameterSpec`:

```
PBEParameterSpec paramSpec = new PBEParameterSpec(salt,
iterations);
Cipher cipher = Cipher.getInstance("PBEWithSHAAndTwofish-CBC");
cipher.init(Cipher.ENCRYPT_MODE, theKey, paramSpec);
```

Nestes casos foram utilizado o algoritmo PBEWithSHAAndTwofish-CBC, o que especifica o PBE utilizando SHA como message digest e Twofish em modo CBC como algoritmo de cifragem. Existe um grande número de possíveis algoritmos de PBE, incluindo:

- PBEWithMD5AndDES (este foi o utilizado no projeto, sem critério para escolha);
- PBEWithSHAAndBlowfish;
- PBEWithSHAAnd128BitRC4;
- PBEWithSHAAndIDEA-CBC;
- PBEWithSHAAnd3-KeyTripleDES-CBC;

A codificação da decifragem segue a mesma linha da cifragem, mudando apenas na inicialização da cifra:

```
cipher.init(Cipher.DECRYPT_MODE, theKey, paramSpec);
```

Os métodos de cifragem e decifragem completos para o SMC, utilizando um contador de iteração de valor mil e um salt de oito bytes gerado randomicamente e adicionado ao início do texto de cifrado, encontram-se na classe `toCipher` apresentada no Apêndice A juntamente com as outras três classes de que consistiu a implementação deste projeto. O número de linhas de código não comentado totalizou 449.

3.6 Dinâmica de funcionamento do SMC

Nesta sessão vamos demonstrar o funcionamento da aplicação e também partes importantes da codificação do programa. A Figura 10 apresenta o Diagrama de Casos de Uso do sistema, que consiste de um ator o qual representa os usuários e três casos de uso relacionados: entrada no grupo, definição de uma cifra para a sessão e saída do grupo.

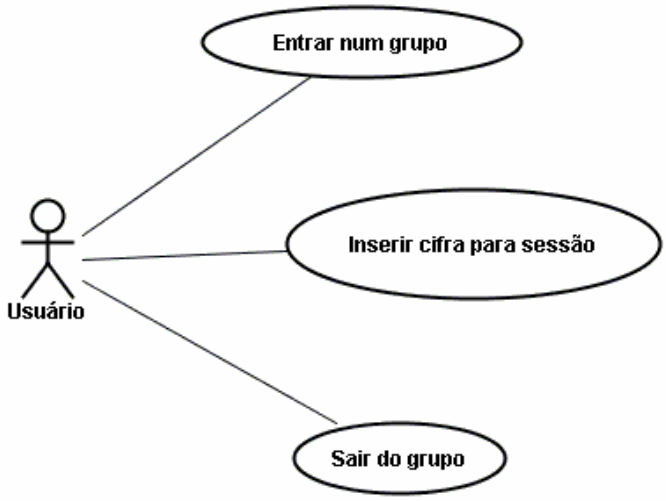

Figura 10. Diagrama de Casos de Uso do SMC

A Figura 11 mostra a tela inicial do SMC, onde o usuário definirá os campos do grupo onde deseja se associar, juntamente a uma porta e um apelido de sua escolha. Fica em exibição o botão de conexão e à direita a lista de usuários que a princípio está vazia. O campo de digitação fica desabilitado, visto que o usuário não está associado a nenhum grupo e fica então proibido de enviar mensagens.

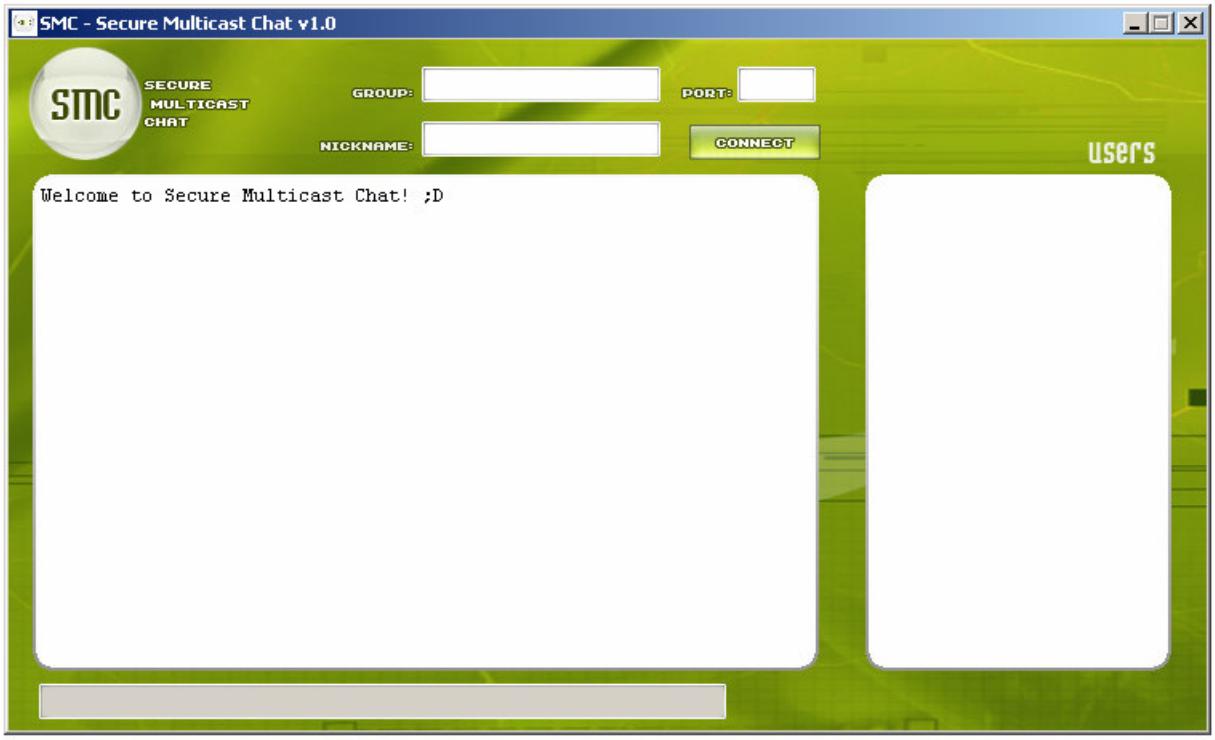

Figura 11. Tela inicial do SMC.

Tendo digitado um IP de Classe D válido e escolhido uma porta e um apelido o usuário, ao clicar no botão de conectar depara-se com uma caixa de diálogo que pede que o mesmo insira a cifra que será utilizada para a cifragem e decifragem dos dados, como mostrado na Figura 12. Caso o usuário cancele ou deixe em branco, a cifra será um espaço em branco.

Se o usuário entra no grupo com sucesso uma mensagem é exibida e o nome do usuário é então adicionado à lista de usuários. À medida que outros usuários vão entrando no grupo utilizando o SMC e a mesma chave de cifragem, mensagens são enviadas aos participantes da sessão informando os novos usuários que entraram na sessão. A entrada do usuário com sucesso e a entrada de um novo usuário no chat são mostradas na Figura 13.

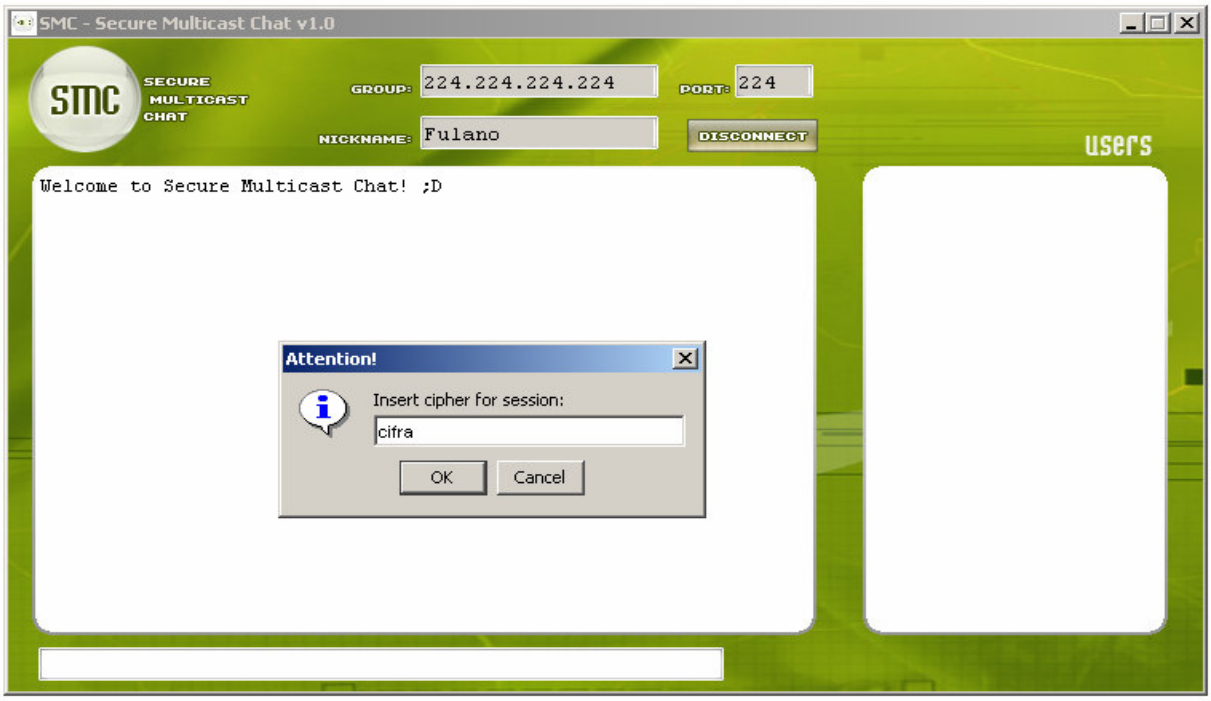

Figura 12. Entrada da senha-cifra.

Figura 13. Usuário conectado e entrada de novo usuário.

Caso o usuário insira um IP de grupo inválido ou tenha deixado algum campo em branco, uma mensagem de erro é exibida e o usuário poderá tentar re-editar os campos para conectar com sucesso. A Figura 14 mostra a tentativa e erro do usuário.

Figura 14. Tentativa e erro de conexão.

A Figura 15 exibe o um caso em que dois usuários se conectam com sucesso em um grupo numa determinada porta e usando a mesma cifra na sessão. Caso um desses fosse diferente, os usuários estariam em sessões diferentes e independentes.

Figura 15. Usuários no mesmo grupo, porta e usando a mesma cifra.

Todas as mensagens trocadas no SMC são devidamente pré-formatadas de forma que a aplicação possa fazer um tratamento das mesmas e verificar se não é apenas um ruído ou

mensagem enviada por um participante do grupo que não está usando o algoritmo e a chave apropriados. Além disso serve para a troca de mensagens de controle, como requerer a lista de usuários que estão na sessão e a forma como o apelido deve ser enviado. Podemos exemplificar os três formatos de mensagem usados no projeto mostrando o que ocorre desde quando um usuário entra na sessão. O mesmo envia um comando "USERS" para o grupo. Ao receber uma mensagem desse tipo, os participantes da sessão enviarão mensagens no formato "NICKADD APELIDO". Ao receber uma mensagem nesse formato, os outros participantes da sessão irão adicionar este usuário caso não o tenham na lista de participantes. Iniciada a sessão e atualizadas as listas, os usuários trocam mensagens no formato "MSG <APELIDO> MENSAGEM". O tratamento dessa mensagem é a exibição na tela das mensagens recebidas. Ao desconectar, o usuário envia um comando explícito "NICKDELETE APELIDO" para que seja removido das listas dos usuários, ou seja, o sistema nesse caso não pode tratar casos extra de travamento do computador ou queda de energia, contando com o funcionamento normal e envio explícito de remoção na lista.

3.7 Comparativo com chats tradicionais

A maioria dos chats tradicionais, a exemplo de IRC e Instant Messengers como como MSN Messenger utilizam uma arquitetura cliente-servidor. Isso significa dizer que de alguma forma, essas aplicações dependem de um servidor central no mínimo para que a conexão entre usuários seja estabelecida e em alguns casos, toda a troca de mensagens necessariamente deve passar por um servidor. Já o Secure Multicast Chat dispensa, a princípio, um servidor para controle, visto que este papel é de certa forma substituído pelos roteadores e os protocolos de grupo e roteamento Multicast. Isto leva a algumas conseqüências e pode ser vantagem de um ponto de vista, mas de outro pode ser desvantajoso:

- Na arquitetura cliente-servidor, todo o tráfego trocado entre os participantes passa pelo servidor. Isso significa que o servidor deve ter um alto poder computacional para tratar de muitas mensagens simultâneas, além de uma grande largura de banda para receber e repassar o tráfego com rapidez. Um servidor deste tipo, juntamente com uma boa conexão de internet significa um alto custo para se ter e manter um serviço num arquitetura deste tipo. O SMC torna-se barato nesse aspecto por aproveitar os roteadores na Internet, tendo também uma maior tolerência a falhas, visto que caso um roteador falhe ou haja falha no cabeamento, as tabelas podem se reorganizar para fazer um novo trajeto dos dados, tendo menos problemas com disponibilidade. O cliente do SMC não necessitará de uma conexão tão boa já que o tráfego que irá receber será apenas o dos grupo que o mesmo estiver participando e caso ele precise enviar uma mensagem para mais de um usuário, a replicação das mensagens fica por conta dos roteadores;
- Em se tratando de funcionalidade, o SMC funciona como um canal, mais conhecido como sala, em um servidor de IRC ou conversas em grupo no MSN, porém utilizando criptografia e sem a possibilidade de conversas privadas. Assim como se poderia implementar criptografia no IRC, pode-se implementar conversas privadas em Multicast, mas na camada de aplicação, ou seja, todos os participantes do grupo receberão a mensagem que deverá ser pré-formatada de forma que na camada aplicação esta seja tratada para ser exibida apenas ao destino correto. Essa não é uma forma conveniente de se implementar uma comunicação privada na Internet, sendo então uso de sockets Unicast mais apropriado;
- Em relação à segurança vemos que o SMC está vulnerável a alguns problemas inerentes da arquitetura atual de Multicast, como a falta de controle de participantes da sessão,

apesar que participantes não autorizados receberão o tráfego totalmente criptografado, mas estes podem gerar ruídos e ataques de DoS que podem prejudicar o desempenho tanto do chat como do próprio sistema operacional dos usuários. A forma de controlar isso seria através do uso de firewalls nos usuários, porém esta não é uma solução plausível já que não se espera que os usuários do chat se preocupem com o gerenciamento de tráfego não autorizado no grupo. O uso de firewall numa arquitetura cliente-servidor é mais simples, pois pode ser centralizado no servidor e aos cuidados de um administrador de sistemas.

- Um problema encontrado na arquitetura do SMC trata-se de autenticidade. Mensagens não cifradas no algoritmo com a chave correta são descartadas pela aplicação. Porém mensagens cifradas capturadas e reproduzidas por um participante do grupo podem chegar à aplicação como se tivessem sido enviadas por um real participante da sessão. Basta que um sujeito copie a mensagem totalmente cifrada e envie para o endereço do grupo.

3.8 Conclusão

Este capítulo tratou do desenvolvimento da aplicação Secure Chat Multicast. Primeiramente dando uma visão geral, os requisitos e a modelagem de casos de uso. Mais adiante explicou as tecnologias utilizadas no projeto, como Java e sockets Multicast, além dos pacotes do JCE para a criptografia. Ao final mostrou a dinâmica do funcionamento do chat e fez um comparativo com os chats cliente-servidor tradicionais. Pudemos ver que o SMC se mantém vulnerável a problemas inerentes do Multicast e que o uso de criptografia sem uma autenticação forte ainda não resolve o problemade troca de dados de forma segura. Logo, podemos ver que a cifragem apenas não é suficiente e que deve-se ter uma forma de autenticar os usuários da sessão individualmente e sem possibilidade de replicação de mensagens por parte dos outros participantes da sessão. Isso pode ser feito através do uso de certificados digitais.

Capítulo 4

Conclusões, Contribuições e Trabalhos Futuros

Neste capítulo, são apresentadas as principais conclusões deste trabalho de pesquisa, suas contribuições e algumas propostas de trabalhos futuros.

4.1 Conclusões

O presente trabalho apresentou como fundamentação teórica os conceitos básicos de Multicast e seu uso na Internet através do MBone, os protocolos de roteamento mais usados como DVMRP, MOSPF, PIM-DM, PIM-SM e CBT, além do protocolo para gerenciamento de grupos, o IGMP. A seguir, abordou as questões de segurança apresentando os aspectos falhos no modelo atual da tecnologia nas camadas de transporte, rede e aplicação.

Mais adiante, mostrou a modelagem e desenvolvimento da aplicação do Secure Multicast Chat que visa a segurança na comunicação em uma sessão de bate-papo Multicast ao utilizar criptografia simétrica, de modo que participantes da sessão que não utilizem a mesma chave e algoritmo de cifragem e decifragem não tenham acesso ao texto claro transmitido. Por fim, foi mostrada a aplicação em funcionamento e foi feito um comparativo do SMC com chats tradicionais cliente-servidor, mostrando que se uma rede tem suporte a Multicast pode oferecer algumas vantagens, como dispensar a utilização de um servidor e por isso pode ser financeiramente uma solução mais barata. Por outro lado pode ser desvantajoso em alguns casos, já que a aplicação continua às falhas de segurança da camada de rede e transporte inerentes ao modelo Multicast atual e pelo fato da possibilidade de captura e replicação de mensagens por usuários não autorizados, mesmo não conhecendo o texto claro nem a chave já que a participação nos grupos é irrestrita. Logo, não é suficiente aplicar apenas a criptografia simétrica como uma forma de autenticação plausível. Em termos de usabilidade, o SMC é comparável a uma sala de IRC ou chat de grupo usando MSN.

Pudemos ver que existem problemas estruturais de segurança no modelo Multicast que não podem ser resolvidos na camada de aplicação de forma aceitável. Isso decorre do fato de que o projeto e implementação inicial do modelo proposto por Steve Deering não levou em consideração essas questões.

4.2 Contribuições e trabalhos futuros

A explanação do modelo Multicast, assim como o alerta para as questões de segurança podem vir a estimular a comunidade científica em busca de soluções estruturais para esses problemas. A popularização de aplicações utilizando este modelo depende bastante destas mudanças.

O SMC, pode adquirir um potencial para o desenvolvimento de novas soluções na camada de aplicação, como aplicação de videoconferência, aliada a autenticação utilizando certificados digitais.

Avaliações comparativas de desempenho utilizando Unicast e Multicast e os diversos algoritmos de criptografia também seria uma sugestão de trabalho futuro.

Obviamente estas soluções na camada de aplicação ainda ficam suscetíveis aos problemas de segurança e escalabilidade do modelo estrutural atual das camadas inferiores e, portanto, dependentes de soluções para os problemas inerentes a esse modelo.

Pesquisas podem então ser feitas nas camadas de rede, transporte e aplicação, tratando dos problemas de escalabilidade, segurança, uso de protocolos orientados a conexão no ambiente Multicast ou aplicações que aproveitem as vantagens da tecnologia.

Bibliografia

[1] ADAMS, A., NICHOLAS, J., SIADAK W. *Draft-ietf-pim-dm-new-v2-03 : Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*, 2003

[2] ALMEROTH, K., AMMAR, M. *Collection and modeling of the join/leave behavior of multicast group members in the MBone*, in: High Performance Distributed Computing Focus Workshop (HPDC'96), Syracuse, New York, 1996.

[3] BALLARDIE, A. *RFC 2189 : Core Based Trees Multicast Routing (CBTv2)*, 1997

[4] BALLARDIE, A. *RFC 2201 : Core Based Trees (CBT) Multicast Routing Architecture*, 1997

[5] BALLARDIE, T., CROWCROFT J. *Multicast-specific security threats and counter-measures*, in: Proceedings of the Symposium on Network and Distributed System Security, San Diego, California, 1995

[6] BAT, T., CHANDRA, R., KATZ D., REKHTER, Y. *RFC 2283 : Multiprotocol Extensions for BGP-4 (MBGP)*, 1998

[7] CAIN, B., DEERING, S., KOUVELAS, I., FENNER, B., THYAGARAJAN, A. RFC 3376 : *Internet Group Management Protocol, Version 3*, 2002

[8] CARONNI, G., WALDVOGEL, M., SUN, D., PLATTNER, B. *Efficient security for large and dynamic groups*, Technical Report TIK Technical Report No. 41, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, 1998.

[9] COSTA, L., DUARTE, O. *Roteamento Multicast na Internet.*

[10] DEERING, S. *RFC 1112 : Host Extensions for IP Multicasting*, 1989

[11] DOBBERTIN, H., Cryptanalysis of MD5 Compress, 1996

[12] ESTRIN, D., FARINACCI, D., HELMY, A., THALER, D., DEERING, S., HANDLEY, M., JACOBSON, V., LIU, C., SHARMA, P., WEI, L. *RFC 2362 : Protocol Independent Multicast Sparse Mode (PIM-SM: Protocol Specification)*, 1998

[13] FENNER, W. *RFC 2236 : Internet Group Management Protocol (IGMPv2)*, 1997

[14] FENNER, B., HANDLEY, M., HOLBROOK, H., KOUVELAS I. *Draft-ietf-pim-sm-v2-new-08 : Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)*, 2003

[15] GONG, L., Shacham, N. *Elements of trusted multicasting, in: Proceedings of the IEEE International Conference on Network Protocols*, 1994

[16] HARDJONO, T., DONDETI, L.R. *Multicast and Group Security* (Artech House Computer Security Series), 2003

[17] HARDJONO, T., WEIS, B. *RFC 3740 : The Multicast Security Architecture*, 2004

[18] HORSTMANN, C., CORNELL G. *Core Java(TM) 2, Volume I Fundamentals (7th Edition) (Core Java 2)*, 2004

[19] HORSTMANN, C., CORNELL G. *Core Java(TM) 2, Volume II Advanced Features (7th Edition)*, 2004

[20] INC. CISCO SYSTEMS, *Interdomain Multicast Solutions Guide*, 2002

[21] KUROSE, JAMES F., ROSS, KEITH W. *Redes de Computadores e a Internet*, 2003
[22] MILLER, C.K. *Multicast Networking and Applications*, Addison Wesley, 1998.
[23] McClure, S. *Hacking Exposed: Network Security Secrets & Solutions*, Fourth Edition (Hacking Exposed) et al, 2003
[24] MCNAB, C. *Network Security Assessment*, 2004
[25] MEYER, D. *RFC 2365 : Administratively Scoped IP Multicast*, 1998.
[26] MOY, J. *RFC 1584 : Multicast Extensions to OSPF (MOSPF)*, 1994
[27] OBRACZKA, K. *Multicast transport protocols: a survey and taxonomy*. IEEE Communications Magazine 36 1, 1998
[28] PELTIER, T.R. *Information Security Policies, Procedures, and Standards: Guidelines for Effective Information Security Management*, 2001
[29] RSA DATA SECURITY, INC. *PKCS #5: Password-Based Encryption Standard. Version 1.4*, June 1991.
[30] SHIROSHITA, T., TAKAHASHI, O., YAMASHITA, M. *Integrating layered security into reliable multicast*, in: Proceedings of the Third International Workshop on Protocols for Multimedia Systems, Madrid, 1996.
[31] STALLINGS, W. *Network and Internetwork Security*, Prentice-Hall, 1995.
[32] TANENBAUM, A.S. *Computer Networks*, Fourth Edition, 2002
[33] WAITZMAN, D., PARTRIDGE, C., DEERING, S. *RFC 1075 : Distance Vector Multicast Routing Protocol*, 1988
[34] WEISS, J. *Java Cryptography Extensions : Practical Guide for Programmers*, 2004
[35] WITTMANN, R., ZITTERBART, M. *Multicast Communication: Protocols, Programming, and Applications* (Morgan Kaufmann Series in Networking), 2002

Apêndice A

CLASSE: **SMC.java**

```
import javax.swing.UIManager;
import java.awt.*;

public class SMC {
     private boolean packFrame = false;
         //Constrói a aplicação
         public SMC() {
         Front frame = new Front();

         if (packFrame) {
           frame.pack();
         }
         else {
           frame.validate();
         }
         //Centraliza a janela
          Dimension screenSize =
          Toolkit.getDefaultToolkit().getScreenSize();
         Dimension frameSize = frame.getSize();
         if (frameSize.height > screenSize.height) {
           frameSize.height = screenSize.height;
         }
         if (frameSize.width > screenSize.width) {
           frameSize.width = screenSize.width;
         }
     frame.setLocation((screenSize.width - frameSize.width) / 2,
     (screenSize.height - frameSize.height) / 2);
         frame.setVisible(true);
       }
     //Método Main
       public static void main(String[] args) {
         try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName(
));
           Front.setDefaultLookAndFeelDecorated(true);
         }
         catch(Exception e) {
           e.printStackTrace();
         }
         new SMC();
       }
}
```

CLASSE: **Front.java**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.StringTokenizer;

public class Front extends JFrame {

     private JPanel contentPane;
     private JLabel backGround = new JLabel();

     private JList userList;
     private DefaultListModel listModel;
     private JScrollPane listScroller = new
     JScrollPane(userList);

     private JTextArea panelChat = new JTextArea();
     private JScrollPane rollChat = new JScrollPane(panelChat);
     private JFormattedTextField boxNick = new
     JFormattedTextField();
     private JFormattedTextField boxGroup = new
JFormattedTextField();
     private JFormattedTextField boxPort = new
JFormattedTextField();
     private JFormattedTextField boxMensagem = new
JFormattedTextField();
     private String chat = "Welcome to Secure Multicast Chat!";
       private String nickName = "";
       private String message = "";
       private String groupIp = "";
       private String portNumber = "";
       private String password = "";

       private JButton buttonConnect = new JButton();
       private JButton buttonDisconnect = new JButton();
       private JButton buttonSend = new JButton();
       private StringBuffer bufferMessage = new StringBuffer("");


       private Connect connect;


       //Constrói o frame
       public Front() {
         enableEvents(AWTEvent.WINDOW_EVENT_MASK);
         try {
           jbInit();
         }
```

```
          catch(Exception e) {
            e.printStackTrace();
          }
        }
        //Inicialização de componentes
        private void jbInit() throws Exception  {

this.setIconImage(Toolkit.getDefaultToolkit().createImage("icone-
smc.jpg"));
          this.setResizable(false);
          this.setSize(new Dimension(754, 455));
          this.setTitle("SMC - Secure Multicast Chat v1.0");

          listModel = new DefaultListModel();

          userList = new JList(listModel);

userList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

          backGround.setBounds (-2, -5, 760, 441);
          backGround.setIcon(new ImageIcon("smc-fundo.jpg"));

          rollChat.setBounds (20, 88, 479, 300);
          rollChat.getViewport().setBackground(Color.white);

rollChat.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCRO
LLBAR_NEVER);
          rollChat.setBorder(null);

          listScroller.setBounds (537, 95, 187, 287);

listScroller.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_
SCROLLBAR_NEVER);
          listScroller.setAutoscrolls(true);
          listScroller.setBorder(null);

          boxMensagem.setBounds (21, 403, 425, 18);
          boxMensagem.setFont(new java.awt.Font("Monospaced", 0,
12));
          boxMensagem.setBorder(null);
          boxMensagem.setEditable(false);

          boxMensagem.addActionListener(new
java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
                              try {
                                   buttonSend_actionPerformed(e);
                              } catch (Exception e2) {
                                   e2.printStackTrace();
                              }
```

```
            }
          });

          panelChat.setFont(new java.awt.Font("Monospaced", 0,
12));
          panelChat.setEditable(false);
          panelChat.setText("" + chat);
          panelChat.setLineWrap(true);

          boxNick.setBounds (260, 53, 144, 18);
          boxNick.setFont(new java.awt.Font("Monospaced", 0, 13));
          boxNick.setBorder(null);
          boxNick.setText("" + nickName);

          boxNick.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                        try {
                             buttonConnect_actionPerformed(e);
                        } catch (Exception e2) {
                             e2.printStackTrace();
                        }
            }
          });

          boxGroup.setBounds (260, 19, 144, 18);
          boxGroup.setFont(new java.awt.Font("Monospaced", 0,
13));
          boxGroup.setBorder(null);
          boxGroup.setText("" + nickName);

          boxPort.setBounds (457, 19, 44, 18);
          boxPort.setFont(new java.awt.Font("Monospaced", 0, 13));
          boxPort.setBorder(null);

          userList.setBounds (537, 95, 187, 287);
          userList.setBackground (Color.white);
          userList.setFont(new java.awt.Font("Monospaced", 1,
12));

          buttonConnect.setBounds (425, 53, 82, 22);
          buttonConnect.setBorder(null);
          buttonConnect.setIcon(new ImageIcon("bt-connect.jpg"));
          buttonConnect.setPressedIcon(new ImageIcon("bt-connect-
over.jpg"));
          buttonConnect.setRolloverIcon(new ImageIcon("bt-
connect.jpg"));
          buttonConnect.setRolloverSelectedIcon(new ImageIcon("bt-
connect.jpg"));
```

```
        buttonConnect.setSelectedIcon(new ImageIcon("bt-
connect.jpg"));

        buttonConnect.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
                                try {

    buttonConnect_actionPerformed(e);
                                } catch (Exception e2) {
                                     e2.printStackTrace();
                                }
          }
        });

        buttonDisconnect.setBounds (425, 53, 82, 22);
        buttonDisconnect.setBorder(null);
        buttonDisconnect.setVisible (false);
        buttonDisconnect.setIcon(new ImageIcon("bt-
disconnect.jpg"));
        buttonDisconnect.setPressedIcon(new ImageIcon("bt-
disconnect-over.jpg"));
        buttonDisconnect.setRolloverIcon(new ImageIcon("bt-
disconnect.jpg"));
        buttonDisconnect.setRolloverSelectedIcon(new
ImageIcon("bt-disconnect.jpg"));
        buttonDisconnect.setSelectedIcon(new ImageIcon("bt-
disconnect.jpg"));

        buttonDisconnect.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e){
         try {
         buttonDisconnect_actionPerformed(e);
         } catch (Exception e3) {
         e3.printStackTrace();
         }
              }
        });

        buttonSend.setBounds (470, 401, 52, 22);
        buttonSend.setVisible (false);
        buttonSend.setBorder(null);
        buttonSend.setIcon(new ImageIcon("bt-send.jpg"));
        buttonSend.setPressedIcon(new ImageIcon("bt-send-
over.jpg"));
        buttonSend.setRolloverIcon(new ImageIcon("bt-
send.jpg"));
        buttonSend.setRolloverSelectedIcon(new ImageIcon("bt-
send.jpg"));
```

```
          buttonSend.setSelectedIcon(new ImageIcon("bt-
send.jpg"));

          buttonSend.addActionListener(new
java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
                try {
                buttonSend_actionPerformed(e);
                } catch (Exception e2) {
                e2.printStackTrace();
                              }
          }
          });

          contentPane = (JPanel) this.getContentPane();
          contentPane.setLayout(null);
          contentPane.setBackground(Color.white);
          contentPane.add(rollChat);
          contentPane.add(listScroller);
          contentPane.add(boxNick);
          contentPane.add(boxGroup);
          contentPane.add(boxPort);
          contentPane.add(buttonConnect);
          contentPane.add(buttonDisconnect);
          contentPane.add(boxMensagem);
          rollChat.getViewport().add(panelChat, null);
          listScroller.getViewport().add(userList, null);
          contentPane.add(buttonSend);
          contentPane.add(backGround);
        }
        //Eventos a executar antes que a aplicação seja fechada
        protected void processWindowEvent(WindowEvent e) {
          super.processWindowEvent(e);
          if (e.getID() == WindowEvent.WINDOW_CLOSING) {
           try {
                Connect.send("NICKDELETE " + getNickName());
                System.exit(0);
                } catch (Exception e1) {
                }
          }
        }

        public String getNickName() {
          nickName = boxNick.getText();
          return nickName;
        }

        public String getGroup() {
          this.groupIp = this.boxGroup.getText();
          return this.groupIp;
```

```
}

public String getPort() {
  this.portNumber = this.boxPort.getText();
  return this.portNumber;
}

public String getPassword() {
  return this.password;
}

public void addUser(String user) {
listModel.addElement(user);
}

public void delUser(String user) {
     listModel.removeElement(user);
}

public JButton getButtonConnect() {
  return this.buttonConnect;
}

public JButton getButtonDisconnect() {
  return this.buttonDisconnect;
}

public JButton getButtonSend() {
  return this.buttonSend;
}

public JTextField getBoxNick() {
  return boxNick;
}

public JTextField getBoxGroup() {
  return this.boxGroup;
}

public JTextField getBoxPort() {
  return this.boxPort;
}

public JTextField getBoxMensagem() {
  return this.boxMensagem;
}


public void setMessage(String message) {
   if(message!=null){
```

```
          this.bufferMessage.append(message + "\n");
          this.panelChat.setText(bufferMessage.toString());

     panelChat.setCaretPosition(panelChat.getDocument().getLength
());
          }
          }

       void buttonConnect_actionPerformed(ActionEvent e) throws
Exception {
         nickName = boxNick.getText();
         groupIp = this.boxGroup.getText();
         portNumber = this.boxPort.getText();
         StringTokenizer nickOk = new StringTokenizer(nickName, "
");
         StringTokenizer groupOk = new StringTokenizer(groupIp,
".");
         StringTokenizer portOk = new StringTokenizer(portNumber,
" ");

         if(nickOk.countTokens()!= 1 || portOk.countTokens()!= 1
|| groupOk.countTokens()!= 4) {
         bufferMessage.append("Error: Verify Group and Nickname
configurations and try again.\n");
         panelChat.setText(bufferMessage.toString());
         }
         else {
           buttonConnect.setVisible (false);
           buttonDisconnect.setVisible (true);
           this.getBoxNick().setEditable(false);
           this.getBoxGroup().setEditable(false);
           this.getBoxPort().setEditable(false);
            this.getBoxMensagem().setEditable(true);
           this.password = "" +
JOptionPane.showInputDialog(null,"Insert cipher for
session:","Attention!",JOptionPane.INFORMATION_MESSAGE);
           buttonSend.setVisible (true);
           bufferMessage.append("Connecting...\n");
           panelChat.setText(bufferMessage.toString());
           boxMensagem.requestFocus();
           this.connect = new Connect(this);
           }
            }

       void buttonDisconnect_actionPerformed(ActionEvent e)
throws Exception {
         buttonDisconnect.setVisible (false);
         buttonConnect.setVisible (true);
         buttonSend.setVisible (false);
         this.getBoxNick().setEditable(true);
```

```
          this.getBoxGroup().setEditable(true);
          this.getBoxPort().setEditable(true);
          this.getBoxMensagem().setEditable(false);
          Connect.send("NICKDELETE " + getNickName());
          Connect.leave();
           bufferMessage.append("Disconnected.\n");
           panelChat.setText(bufferMessage.toString());
           this.listModel.clear();
        }

        void buttonSend_actionPerformed(ActionEvent e) throws
Exception{
          this.message = boxMensagem.getText();

          StringTokenizer msgOk = new StringTokenizer(message, "
");
          if(msgOk.countTokens () == 0) {
           boxMensagem.setText("");
          }
          else {
           boxMensagem.setText("");
           boxMensagem.requestFocus();
           Connect.send("MSG " + this.getNickName() + " " +
message);
          }
        }
        //Método de processamento de comandos
        public void ProcessaComando(String command, String second,
String message) throws Exception{

           if (command == null){
                return;
           }
           if (command.equals("USERS")){
                Connect.send("NICKADD " + getNickName());
           }else if (command.equals("NICKADD") &&
!listModel.contains(second)) {
                addUser(second);
                setMessage (second + " has joined the session.");
           }else if (command.equals("MSG")){
                setMessage("<" + second + ">" +
message.substring(command.length()+second.length()+1));
           }else if (command.equals("NICKDELETE")){
                delUser(second);
                setMessage (second + " has left the session.");
           }

           return;
     }
     }
```

CLASSE: **Connect.java**

```
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

class Connect implements Runnable {

        private String nick;
        private static InetAddress group;
        private static int port;
        private static String password;
        private String command = null;
        private String second = null;
        private Front front = new Front();
        private Thread receive;
       private DatagramPacket recv;
       private static MulticastSocket msocket;
       private String message = null;
       private StringTokenizer text;

public Connect(Front front) throws Exception
{

    try {
     this.front=front;
     //   Converte o número da porta para integer.
        port = Integer.parseInt(front.getPort());
        group = InetAddress.getByName(front.getGroup());
        password = front.getPassword();

        //  Cria o socket Multicast
        msocket = new MulticastSocket(port);;

        //     Seta o TTL para 32
        msocket.setTimeToLive(32);

        //     Reuso do endereço
        msocket.setReuseAddress(true);

        //     Entra no grupo
        msocket.joinGroup(group);

        front.setMessage("Joined Group: " + group + " Port: " +
port);

        send("USERS");

       }
       catch(IOException ioe) {
```

```
        this.front.setMessage("Error: Verify Group and Nickname
configurations and try again.\n");
        front.getBoxNick().setEditable(true);
        front.getBoxGroup().setEditable(true);
        front.getBoxPort().setEditable(true);
        front.getButtonConnect().setVisible(true);
        front.getButtonDisconnect().setVisible(false);
        front.getButtonSend().setVisible(false);
      }
      receive = new Thread(this);
      receive.start();
}


public void run()
{
       byte[] clearText= new byte[1000];
       String tmp;

       while (!msocket.isClosed())
       {
       try{
               for(;;)
               {
                    try{
                          //Trata os dados recebidos.
                          recv = new DatagramPacket(clearText,
clearText.length);
                          msocket.receive(recv);
                          tmp =
toCipher.Decrypt(password.toCharArray(),new
String(recv.getData(),0,recv.getLength())).toString();

                          text = new StringTokenizer(tmp, " ");

                          command = text.nextToken();

                          if (text.hasMoreTokens())
                          second = text.nextToken();

                          front.ProcessaComando(command, second
,tmp);

                    }    catch (Exception e) {
                     break;
                }

               }
       } catch (Exception e) {
              e.printStackTrace();
```

```
        }
      }
}

public static void send(String mensagem) throws Exception {
    try {
          byte[] clearText =
toCipher.Encrypt(password.toCharArray(),mensagem).getBytes();

            DatagramPacket dp = new DatagramPacket(clearText,
clearText.length, group, port);

            //Envia o pacote para o grupo
            msocket.send(dp);

    }
    catch(IOException ioe) {
    }
  }


public static void leave() throws IOException{
        //Sai do grupo
    msocket.leaveGroup(group);
    //Fecha o socket
    msocket.close();
}

}
```

CLASSE: **toCipher.java**

```
import java.util.Random;
import javax.crypto.*;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;

import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;


public class toCipher {

//Contador de iterações

private static int ITERATIONS = 1000;

public static String Encrypt(char[] password, String plaintext)
throws Exception
{

//Cria um salt randômico de 64 bits (8 bytes)

byte[] salt = new byte[8];
Random random = new Random();
random.nextBytes(salt);

//Cria o PBEKeySpec com a senha fornecida

PBEKeySpec keySpec = new PBEKeySpec(password);

//Pega uma SecretKeyFactory para o PBEWithMD5AndDES

SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance("PBEWithMD5AndDES");

//Cria a chave

SecretKey key = keyFactory.generateSecret(keySpec);

//Cria um ParameterSpec dados o salt e as iterações

PBEParameterSpec paramSpec = new PBEParameterSpec(salt,
ITERATIONS);

//Cria a cifra e inicializa para a cifragem

Cipher cipher = Cipher.getInstance("PBEWithMD5AndDES");
cipher.init(Cipher.ENCRYPT_MODE, key, paramSpec);
```

```
//Executa a real cifragem

byte[] ciphertext = cipher.doFinal(plaintext.getBytes());

BASE64Encoder encoder = new BASE64Encoder();

//Codifica na base 64 os bytes para o salt e texto

String saltString = encoder.encode(salt);

String ciphertextString = encoder.encode(ciphertext);

return saltString+ciphertextString;

}

public static String Decrypt(char[] password, String text) throws
Exception
{
//Divide o texto nos strings de salt e texto a ser decifrado.

String salt = text.substring(0,12);
String ciphertext = text.substring(12,text.length());

//Decodifica na base 64 os bytes para o salt e texto

BASE64Decoder decoder = new BASE64Decoder();
byte[] saltArray = decoder.decodeBuffer(salt);
byte[] ciphertextArray = decoder.decodeBuffer(ciphertext);

//Cria o PBEKeySpec dada a senha

PBEKeySpec keySpec = new PBEKeySpec(password);

//Pega uma SecretKeyFactory para o PBEWithMD5AndDES

SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance("PBEWithMD5AndDES");

//Cria a chave

SecretKey key = keyFactory.generateSecret(keySpec);

//Cria um ParameterSpec dados o salt e as iterações

PBEParameterSpec paramSpec = new PBEParameterSpec(saltArray,
ITERATIONS);
```

```
//Cria a cifra e inicializa para a decifragem

Cipher cipher = Cipher.getInstance("PBEWithMD5AndDES");

cipher.init(Cipher.DECRYPT_MODE, key, paramSpec);

//Executa a real decifragem

byte[] plaintextArray = cipher.doFinal(ciphertextArray);

return new String(plaintextArray);
}

}
```