UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

FCT - Faculdade de Ciências e Tecnologia
DMEC - Departamento de Matemática, Estatística e Computação
Bacharelado em Ciência da Computação

Utilização do protocolo anycast em ambientes virtuais distribuídos visando à escalabilidade do sistema

RICARDO SILVA ALVES DE OLIVEIRA

FCT-Unesp - Presidente Prudente
2011

RICARDO SILVA ALVES DE OLIVEIRA

Utilização do protocolo anycast em ambientes virtuais distribuídos visando à escalabilidade do sistema

Monografia apresentada ao Departamento de Matemática, Estatística e Computação (DMEC), da Faculdade de Ciências e Tecnologia (FCT), UNESP, como parte das atividades da disciplina Trabalho de Conclusão de Curso, necessária para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Ronaldo Celso Messias Correia

Presidente Prudente
2011

Resumo

Ambientes Virtuais Distribuídos (AVDs) permitem que múltiplos usuários interajam com um mundo virtual compartilhando, manipulando objetos. Cada usuário tem sua própria visão de mundo e todas alterações de estado do ambiente são distribuídas entre os demais usuários. Devido as restrições tecnológicas, a maioria dos sistemas de ambientes virtuais distribuídos suportam um número limitado de usuários. As maiores limitações para o desenvolvimento de AVDs de grande escala são impostas pelo sistema de comunicação. Conforme o número de usuários cresce, aumenta a quantidade de banda necessária para a troca de informações em tempo real entre as estações para atualizar o ambiente e mantê-lo em um estado consistente para todos os usuários conectados. Com base nessa situação é proposto o desenvolvimento de um *framework* que utiliza o protocolo *anycast* em nível de aplicação que forneça um balanceamento de carga entre os servidores de região e busque diminuir a latência da entrega de mensagens entre os participantes do AVD selecionando o servidor de região mais adequado para o participante se conectar ao ambiente.

Palavras-chave: *Anycast*, Ambientes Virtuais Distribuídos, escalabilidade.

Abstract

Distributed Virtual Environments (DVE) allow multiple users interact with a virtual world by sharing, manipulating objects. Each user has his/her own vision of this world and every changes of state of the environment are distributed among other users. Due to technological constraints, most systems for distributed virtual environments support a limited number of users. The major limitation for development of DVE scale are imposed by the communication system. As the number of users grows, the amount of bandwidth required to exchange information in real time among stations to upgrade the environment and keep it in a consistent state for all connected users. Based on this situation is proposed a framework that uses the protocol anycast in the application layer that provides a load balancing among servers in the region and aims at reducing the latency of message delivery between participants selecting the DVE server best suited for the region participant to connect to the environment.

Keywords: *Anycast, Distributed Virtual Environments, scalability*

Índice

Lista de Figuras

CAPÍTULO 1

Introdução

Ambientes virtuais distribuídos (AVDs) permitem que múltiplos usuários, mesmo que estejam espalhados geograficamente, interajam com um mundo virtual compartilhando e manipulando objetos. Cada usuário tem sua própria visão deste mundo e toda alteração de estado do ambiente são distribuídas entre os demais usuários.

A variedade de funcionalidades que os AVDs devem apresentar torna esse tipo de sistema cada vez mais complexo à medida que se pretende aproximá-lo dos ambientes reais. Esses ambientes de simulação devem suportar intensa interação distribuída, pois com a participação de diversos usuários o ambiente deve garantir confiabilidade e desempenho satisfatórios para que participantes possam interagir entre si em tempo real, e inclusive reagir a informações do ambiente também em tempo real.(CORREIA, 2005).

1.1 Objetivos e Motivação

As maiores limitações para o desenvolvimento de AVDs de grande escala são impostas pelo sistema de comunicação. Conforme o número de usuários cresce, aumenta a largura de banda necessária para a troca de informações em tempo real entre as estações para atualizar o ambiente e mantê-lo em um estado consistente para todos os usuários conectados.

Se não for projetada cuidadosamente, a rede poderia facilmente se tornar um fator limitante e corromper a sincronização entre usuários (ERASLAN et al., 2007). Reduzindo a carga de rede pode-se diminuir as latências das transmis-

sões causadas pelos congestionamentos e consequentemente, as atualizações de mudança de estado do ambiente serão enviadas mais rapidamente. Desse modo, é proposto a utilização de um novo protocolo de comunicação, protocolo *anycast* buscando superar tais dificuldades.

Anycast (PARTRIDGE; MENDEZ; MILLIKEN, 1993) é um serviço que permite um nó enviar uma mensagem para um membro próximo de um grupo, onde proximidade é definida usando uma métrica. Solicitações de serviços dirigida a este grupo *anycast* são retransmitidas para o servidor mais adequado a atender a solicitação de um determinado cliente. O protocolo *anycast* fornece um modelo versátil e eficaz que permite a aplicação robustez e balanceamento de carga dinâmico.

Este trabalho tem como objetivo principal o desenvolvimento de um *framework* que utilize as vantagens do protocolo *anycast* em nível de aplicação para balanceamento de carga entre os servidores de região e diminuir a latência da troca de pacotes na rede pelos participantes do AVD.

1.2 Organização da Monografia

Dentro do contexto do presente projeto a monografia possui a seguinte estrutura:

- No Capítulo 2 são abordados conceitos e características de ambientes virtuais distribuídos (AVDs). Mostrando as vantagens e as desvantagens nas topologias de comunicação usadas em AVDs, bem como meios de otimização de tráfego pela rede e balanceamento de carga entre os servidores.

- No Capítulo 3 é apresentado o *framework Anycast* desenvolvido por este trabalho. Seu uso, mecanismos, e modelo de desenvolvimento são detalhados nesse capítulo.

- Por fim, no Capítulo 4 são apresentadas as conclusões abragendo o trabalho como um todo e oportunidades para o desenvolvimento de possíveis trabalhos futuros e expondo as considerações finais.

CAPÍTULO 2

Ambientes Virtuais Distribuídos

Ambientes Virtuais Distribuídos (AVDs) são sistemas que permitem múltiplos usuários interajirem em tempo real num ambiente compartilhado, mesmo que os usuários estejam distribuídos geograficamente (SINGHAL; ZYDA, 1999).

AVDs lidam com questões acerca de como interagir com um objeto e até com outros usuários, simulando espaços e atividades reais por meio do uso incorporado de computação gráfica 3D e som estéreo (DUNDES, 2008).

Aplicações de ambientes virtuais distribuídos estão presentes em áreas como educação (cursos à distância, visualização de conteúdo, como superfícies na matemática ou dinâmica de fluidos na física), saúde (treinamento cirúrgicos, anatomia), entretenimento (jogos), arquitetura e engenharia (navegação de maquetes virtuais, auxilio do design de equipamentos), treinamento militar (simulação de combate, simulação de vôo, etc.).

Um AVD é caracterizado pelos seguintes fatores segundo (SINGHAL; ZYDA, 1999):

Sensação de espaço compartilhado: Todos os participantes de um AVD têm a sensação de estarem presentes no mesmo espaço. As características do local devem ser as mesmas para todos os presentes no local.

Sensação de presença compartilhada: Cada participante de um AVD é representado por um avatar, que pode ter qualquer forma. Todos os participantes do AVD podem ver os avatares dos outros participantes que estão presentes.

Sensação de tempo compartilhado: Os participantes podem ver as ações realizadas por outros participantes em tempo real.

Meio de comunicação: Os participantes podem comunicar-se uns com os outros de diversas maneiras, fazendo uso de textos, gestos, som dependendo do grau de realismo do ambiente.

Meio de compartilhamento: Os participantes além de interagir uns com os outros, devem ser capazes de interagir com o próprio ambiente, manipulando e movendo objetos.

Dependendo do grau de realismo que se deseja proporcionar ao ambiente virtual, um AVD pode se tornar algo muito complexo. Perceber a ação de outros participantes em tempo real e poder reagir a elas também em tempo real é uma característica desejada em um AVD e o que torna um sistema complexo de se projetar.

Ao mesmo tempo em que os ambientes virtuais podem ser aplicados nas mais variadas áreas (educação, simulação, visualização científica, jogos), também a sua forma de construção pode variar bastante (VANDEIR, 2001).

Uma das grandes preocupações em projetos de AVDs está na camada de comunicação do sistema, pois provê o mecanismo necessário para o compartilhamento de objetos, mensagens de atualização de estado dos objetos e troca de informações entre os usuários do sistema.

Se não houver um projeto cuidadoso na camada de comunicação, ela pode facilmente se tornar um fator limitante do sistema, corrompendo a sincronização entre os usuários. O projeto de infra-estrutura de comunicação é importante para otimizar o uso de recursos da rede, diminuir a latência e suportar ambientes de larga escala (ERASLAN et al., 2007).

Algumas das exigências de AVDs que a camada de comunicação deve atender:

Baixa latência: Latência é o tempo decorrido entre o momento no qual o pacote deixa a origem e chega ao seu destino. Se a latência for muito alta as ações de um usuário demoram a chegar aos demais usuários, prejudicando a interatividade e a dinâmica do ambiente. Se o objetivo do ambiente virtual é simular um mundo real, precisa operar em tempo real, considerando-se a percepção humana para serem satisfatórias para os usuários (CORREIA, 2005).

Confiabilidade: a confiabilidade garante que todos os pacotes transmitidos serão entregues aos seus destinos corretamente, sem perdas ou corrompimentos, para isso as vezes a retransmissão de alguns pacotes sejam necessárias o que é prejudicial para aplicações de tempo real. Para o projeto de AVDs deve-se analisar qual o impacto do não recebimento de alguns pacotes e aplicar técnicas para que o não recebimento de pacotes não se torne algo muito prejudicial.

Largura de banda: corresponde à quantidade de dados que podem ser transmitidos em um determinado intervalo de tempo. Isso influencia em AVDs pois as ações realizadas por um usuário deve ser enviadas para os demais o mais rápido possível.

Em AVDs geralmente tem-se um número grande de usuários conectados. Para suportar interatividade em tempo-real a escalabilidade do sistema se torna um aspecto chave em AVDs.

Existem cinco pontos que devem ser considerados em sistemas grandes de AVDs escaláveis, a arquitetura de comunicação, o gerenciamento de interesse, o controle de concorrência, a replicação de dados e a distribuição de carga (LEE et al., 2007).

2.1 Topologia de comunicação em AVDs

A topologia de comunicação define como os dados fluirão entre os participantes do AVD (ERASLAN et al., 2007). Assim, a topologia torna-se uma solução básica para a otimização do tráfego de AVDs (JUNIOR, 2000).

Os dados de um AVD trafegam pela rede dentro de PDUs (*Protocol Data Unit*) que são a unidade básica de transmissão de dados em um AVD. Não existe um único um formato padronizado de PDU, há PDUs para cada serviço específico em um AVD (SINGHAL; ZYDA, 1999). A seguir algumas topologias de comunicação em AVDs.

2.1.1 Topologia Cliente-Servidor

Nessa topologia existe um servidor central que mantém todas as informações sobre o ambiente. Todas as PDUs transmitidas no ambiente passam pelo servidor que as encaminham para seus destinos apropriados. O servidor tem a responsabilidade de manter o ambiente consistente.

A vantagem dessa topologia é que facilita a implementação e o servidor pode aplicar alguns filtros para se evitar mensagens redundantes trafegando pela rede.

Com o controle centralizado do ambiente, não existe a possibilidade de surgirem inconsistências dos dados entre os participantes (CORREIA, 2005). Quando um novo usuário se conecta ao ambiente ele recebe do servidor todas as informações do ambiente. O servidor tem a tarefa de incluí-lo na base de dados de usuários e atualizar os demais usuários sobre o novo usuário.

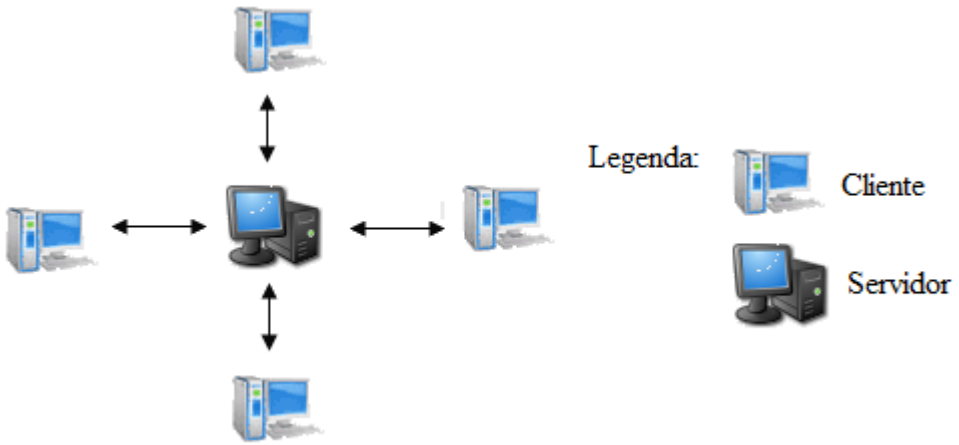

Figura 2.1: Modelo Cliente Servidor

As desvantagens dessa topologia é que o servidor passa a ser um fator limitante do sistema, pois tem que processar cada mensagem enviada no ambiente. Se existir N usuários e todos enviarem mensagens para atualizar suas posições, então deve ser enviadas $N*(N-1)$ mensagens por atualização, o que torna a complexidade do ambiente $O(N^2)$ (ERASLAN et al., 2007).

Outra desvantagem é que as mensagens precisam passar por um nó a mais da rede para chegar aos destinatários, aumentando o tempo de transmissão. O servidor também se torna um único ponto de falha do sistema. Uma quebra do servidor provoca a quebra do ambiente inteiro.

E ainda independentemente da capacidade de processamento do servidor, o número de participantes aceitos em uma estrutura Cliente-Servidor é bastante inferior àquele obtido com a topologia *Peer-to-Peer* (JUNIOR, 2000).

Apesar dos problemas descritos, alguns AVDs existentes adotam a topologia cliente-servidor como sua topologia de comunicação, uma vez que esta topologia é eficaz em manter o sistema consistente e pode ser implantado na atual internet sem a capacidade de *multicast* (LEE et al., 2007).

Para escalabilidade, alguns sistemas introduzem vários servidores para superar o limite de um servidor quanto ao número de clientes e resolver o problema de um único ponto de falha (FUNKHOUSER, 1995).

2.1.2 Topologia *Peer-to-Peer* (P2P)

Essa topologia é caracterizada por permitir a comunicação direta entre qualquer um dos elementos da rede (DUNDES, 2008). O problema de um único ponto de falha nessa topologia não existe. Se por acaso um cliente para de funcionar, os demais continuam interagindo uns com os outros transparentemente (LEE et al., 2007).

A consistência do ambiente é responsabilidade dos clientes que trocam

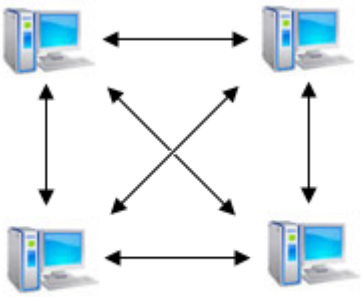

Figura 2.2: Topologia P2P

mensagens de atualizações para os outros clientes. Se a comunicação entre clientes for *unicast*, quanto maior o número de clientes maior o número de conexões que cada cliente deve gerenciar o que pode sobrecarregá-lo, comprometendo a escalabilidade do sistema.

O conjunto de medidas relacionadas à entrada e a saída de usuários de um determinado sistema é bastante complexo, pois cada participante deve estabelecer comunicação com cada outro participante (DUNDES, 2008).

A complexidade do ambiente na topologia P2P utilizando *unicast* é da ordem de $O(N^2)$, sendo N o número de clientes, pois para cada atualização gerada por um cliente são necessárias transmitir $N * (N - 1)$ mensagens para manter a consistência.

Uma rede baseada na topologia *Peer-to-Peer* que utilize comunicação *multicast* faz com que o número de mensagens enviadas para sincronização do ambiente virtual seja bem menor do que uma topologia unicamente *Peer-to-Peer* (JUNIOR, 2000).

Topologia P2P combinada com técnicas de transmissão *multicast* provê eficiência ao sistema evitando duplicatas de pacotes desnecessárias (ERASLAN et al., 2007). Quando um usuário entra no ambiente, ele deve se inscrever em um grupo *multicast* para receber e enviar atualização ao ambiente.

Quando um usuário se move no ambiente, por exemplo, é gerado somente um pacote de atualização que é entregue a todos os demais usuários do grupo *multicast*. Com isso a complexidade do sistema é da ordem de $O(N)$ (ERASLAN et al., 2007), com N sendo o número de usuários do ambiente. A latência também é reduzida quase pela metade (ERASLAN et al., 2007).

Como não existe um servidor central nessa topologia, um novo usuário que se conecta ao ambiente só tem conhecimento de todos os usuários presentes quando estes enviam mensagens de atualização. Os usuários devem gerar constantemente mensagens periódicas de *keep-alive* (constante checagem em relação ao funcionamento normal de um elemento de rede) amenizando o problema (DUNDES, 2008).

Enquanto o *multicast* pode reduzir a sobrecarga de comunicação, ele não

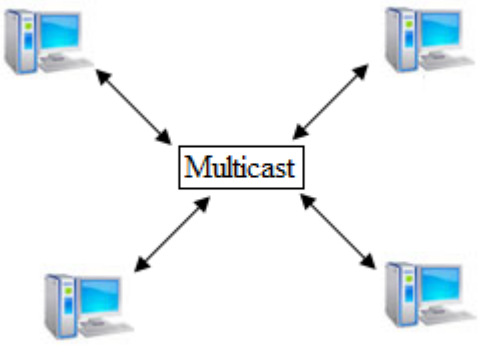

Figura 2.3: Topologia *Peer-to-Peer* com *multicast*

pode aplicar filtragens mais específicas como filtragem por usuário, como na comunicação *unicast* (LEE et al., 2007).

2.1.3 Topologia Híbrida

A topologia hibrida une as duas topologias anteriores, Cliente-Servidor e *Peer-to-Peer*. O uso de Topologias Híbridas contorna o problema típico do uso de um único servidor central (Topologia Cliente-Servidor) e utiliza uma estrutura integrada de comunicação que possibilita realizar a distribuição das conexões dos diversos servidores, acrescentando uma grande escalabilidade ao sistema (ERASLAN et al., 2007).

Essa topologia utiliza um servidor central que mantém informações sobre os usuários no ambiente. Quando um novo usuário se conecta ao ambiente esse servidor se encarrega de passar uma visão consistente do ambiente ao novo usuário. Os usuários trocam mensagens diretamente uns com os outros utilizando *multicast*. O servidor não redistribui mensagens, mas só as recebem e atualiza o estado atual do ambiente (LEE et al., 2007). Nesse caso o servidor se faz necessário para executar um papel de administrador, como por exemplo, no gerenciamento de conexão, autenticação e desconexão de usuários (RODRIGUES et al., s.d.).

Há duas abordagens principais em relação ao emprego de Topologias Híbridas (CAPIN et al., 1999):

1. **Subdivisão do mundo virtual:** o mundo virtual é dividido em regiões, onde cada região é gerenciada por um servidor específico. Os servidores estabelecem a comunicação entre si por meio do *multicast*, e com os clientes também.

   Assim, quando um participante percorrer o ambiente, ele está na verdade se conectando e se desconectando de servidores responsáveis pela administração e gerenciamento de cada região (DUNDES, 2008).

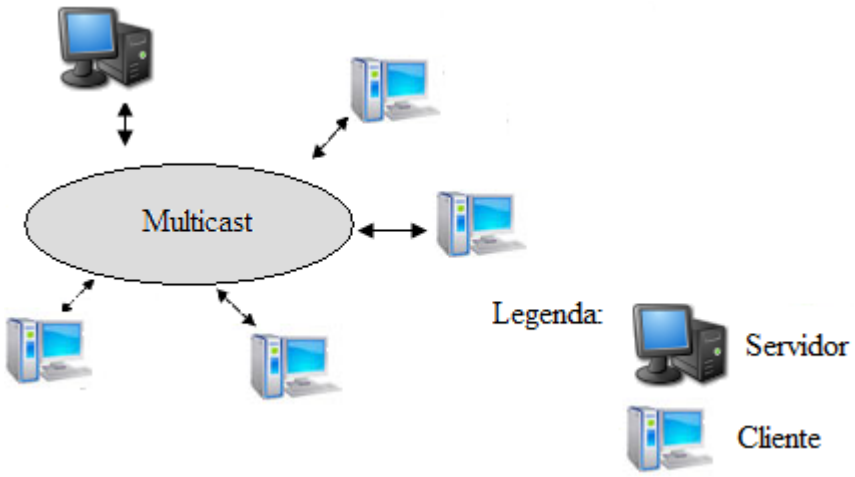

Figura 2.4: Topologia Hibrida

2. **Subdivisão de participantes:** consiste na geração de cópias completas do AVD em vários servidores no ambiente. O participante do AVD tem a liberdade de escolher a qual servidor se conectar, o ideal é o servidor de mais fácil acesso (ERASLAN et al., 2007). Sua grande vantagem é a possibilidade dos servidores apresentarem um número igual de participantes conectados por intermédio de técnicas de balanceamento (DUNDES, 2008).

2.1.4 Mensagens de comunicação

Para que a comunicação entre os usuários e o sistema seja realizada de forma eficiente se faz necessário o uso de protocolos padronizados para troca de mensagens.

Dentre os protocolos disponíveis na pilha de protocolos TCP/IP para esse fim temos:

IP (*Internet Protocol*): é o protocolo da camada de rede da pilha de protocolo TCP/IP que provê comunicação entre *hosts*. O modelo de comunicação do IP é não orientado a conexão, não confiável.

TCP (*Transmission Control Protocol*): é um protocolo da camada de transporte da pilha de protocolos TCP/IP que tem o serviço de entrega de mensagens confiável e orientado a conexão, o que significa que todas as mensagens enviadas à um destino são recebidas.

UDP (*User Datagram Protocol*): também é um protocolo da camada de transporte. Em oposição ao TCP, o UDP é um serviço de entrega de mensagens não confiável e não orientado a conexão. Significa que as mensagens podem não chegar ao seu destino.

Apesar dessa desvantagem, o UDP é muito mais rápido que o TCP e pode ser usado para aplicações em que os pacotes perdidos não tenham tanta importância como na transmissão de áudio e vídeo (DIEHL, 2001).

Além dos protocolos de comunicação, para que ocorram trocas de informações entre os diversos participantes de um AVD faz-se necessário que ocorra alguma padronização nas mensagens de comunicação utilizadas entre os participantes e o ambiente (DUNDES, 2008).

Alguns desses padrões de mensagens são apresentados a seguir:

DIS: o protocolo DIS (*Distributed Interactive Simulation*) consiste em um grupo de padrões desenvolvidos pelo Departamento de Defesa Americano e por entidades da indústria Americana voltados para topologias de comunicação, formato e conteúdo de dados, interação e informação sobre entidades virtuais, gerenciamento de simulações, etc. (SINGHAL; ZYDA, 1999).

O DIS é derivado do SIMNET, um simulador de treinamento militar desenvolvido pelo Departamento de Defesa Americano. Algumas características do DIS herdadas do SIMNET:

Arquitetura objeto-evento: A arquitetura objeto-evento modela o ambiente virtual como uma coleção de objetos. As interações entre estes objetos são realizadas através de um conjunto de eventos, que são mensagens trocadas pela rede indicando uma alteração no estado do objeto (CORREIA, 2005).

Nós de simulação autônomos: São responsáveis por enviar mensagens de atualizações dos objetos periodicamente. O mecanismo de *Dead reckoning*: É utilizado na predição das posições dos objetos, minimizando a troca de mensagens da rede (CORREIA, 2005).

O DIS tem PDUs para cada serviço especifico, como disparo de projeteis, detonação e marcação de ambiente. O PDU mais importante é o ESPDU (*Entity State PDU*) que contem todas as informações sobre um determinado objeto, como localização, velocidade, direção. Uma carência do protocolo DIS é a sua falta de estrutura para suportar a comunicação entre usuários. Isso se deve à sua natureza para fins de simulações militares.

Outra deficiência é que o DIS não é escalável devido o envio de mensagens constantes de atualizações. Foi estimada que, para simulações com 100.000 participantes, é exigida uma conexão de 375 Mbits para cada participante (DIEHL, 2001).

DWTP: O protocolo DWTP (*Distributed Worlds Transfer and Communication Protocol*) foi desenvolvido pela *German National Research Center for Information Tecnology*. O DWTP foi projetado para rodar na camada de aplicação e suportar aplicações heterogêneas enviando dados de diferentes tipos.

O DWTP usa mensagens NACK para avisar que uma mensagem não foi devidamente recebia. Um número pequeno de participantes envia um ACK, ao

receber corretamente um pacote, contendo o número de todos os participantes registrados. Um participante que receber um ACK ao invés do próprio pacote envia um NACK solicitando reenvio. O pacote é enviado novamente por TCP *unicast* para o participante que solicitou (DIEHL, 2001).

***Multi-User 3D protocol*:** O Mu3D (*Multi-User 3D protocol*) foi desenvolvido por Galli e Luo da *University of Balearic Islands*, para um projeto de arquitetura colaborativa. Em contraste com o DWTP, o protocolo Mu3D é *peer-to-peer*; já em contrate com o DIS, ele não envia os dados completos, somente as atualizações (DIEHL, 2001).

Suponha-se, por exemplo, um jogo de tabuleiro distribuído em que só possa haver um objeto em um quadrado por vez. Suponha que a ordenação da origem foi violada: o cliente 2 não recebe a mensagem na ordem em que foi enviada pelo cliente 1. Pode-se forçar a ordenação na origem ao se usar números sequênciais. Se um cliente receber uma mensagem com um número de sequência maior que o último, ele espera até que tenha recebido todas as mensagens intermediárias (RODRIGUES et al., s.d.).

Devido ao fato do Mu3D enviar atualizações e não o estado completo da entidade, o tamanho das mensagens, incluindo todos os *overheads* das camadas inferiores do protocolo, é pequena (200 a 300 bytes) (DIEHL, 2001).

2.2 Gerenciamento de Interesse

Embora a capacidade de computação e a velocidade de renderização esteja aumentando rapidamente, os recursos da rede ainda continuam caros em comparação com recursos computacionais (LEE et al., 2007).

Conforme o número de usuários aumenta no ambiente, aumenta a quantidade de dados trafegando pela rede gerando congestionamentos na rede. Esses congestionamentos podem corromper os dados, provocar atrasos na entrega dos dados o que pode gerar inconsistência entre os usuários e prejudicar a interação entre eles.

Para contornar esses problemas alguns esquemas de gerenciamento de interesse são desenvolvidos. O gerenciamento de interesse leva o fato de que nem todas as mensagens de atualização do ambiente sejam importantes para todos os usuários.

O gerenciamento de interesse pode ser dividido em abordagem baseada em proximidade, abordagem baseada em classe e em uma abordagem hibrida (GREENHALGH; BENFORD, 1997)(SINGHAL; ZYDA, 1999).

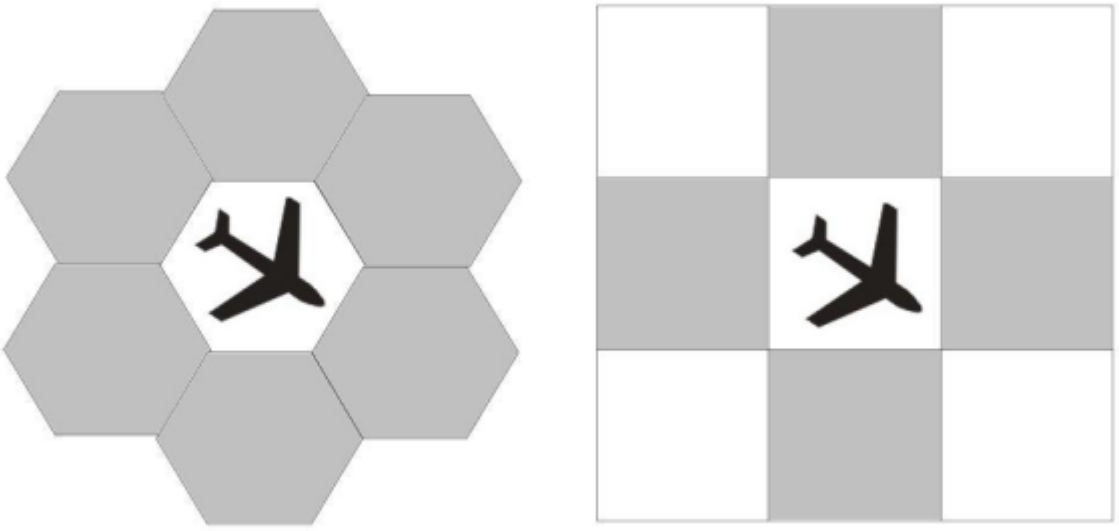

Figura 2.5: Partição geográfica de regiões: Hexagonal (CORREIA, 2005) e Retangular Regular

2.2.1 Gerenciamento de interesse baseado em proximidade

Esse gerenciamento leva em consideração o fato de que o ser humano não pode interagir com objetos que estejam longe dele. Há duas abordagem para se implementar o gerenciamento baseado em proximidade:

- Filtragem baseada em regiões
- Filtragem baseada em aura

2.2.1.1 Filtragem baseada em regiões

Na filtragem baseada em regiões o ambiente é dividido em regiões adjuntas e cada região é associada a um grupo *multicast* gerenciado por um gerente chamado gerente de região.

Um usuário pertencente a uma região interage com outros usuários pertencentes a essa região ou em regiões vizinhas (LEE et al., 2007).

Os critérios de divisão constituem ponto fundamental desta área de pesquisa, podendo o ambiente ser dividido em um formato retangular ou hexagonal (CORREIA, 2005).

A identificação dos componentes pertencentes a regiões retangulares é mais simples que em regiões hexagonais. Por outro lado, regiões hexagonais têm seis regiões vizinhas enquanto regiões retangulares têm oito regiões vizinhas e um componente que se situa em uma zona limite de uma região pode pertencer em até quatro regiões diferentes na divisão retangular em contraste na divisão hexagonal que ele pode pertencer em no máximo três.

No entanto, este pode apenas executar uma filtragem de mensagem grosseira, uma vez que a filtragem é feita com a unidade de uma região (LEE et al., 2007).

2.2.1.2 Filtragem baseada em aura

A aura é a esfera de influência que constitui os dados disponíveis para as entidades interessadas em receber esta informação (RODRIGUES et al., s.d.). Cada usuário do AVD tem uma área limite de interesse, que pode ser o limite da audição e da visão, na qual está apto a receber dados do ambiente. A troca de dados acontece somente entre usuários e objetos do ambiente em que suas auras se interceptam formando um grupo *multicast*.

Isto não só reduz o número de mensagens trocadas no sistema, mas esse método também pode realizar uma filtragem mais fina de mensagens com uma unidade de usuário (LEE et al., 2007).

Por outro lado, exige uma capacidade de processamento muito maior por parte do cliente do sistema, pois precisa detectar colisões continuamente e gerenciar a formação e a destruição de grupos *multicast* dinamicamente.

2.2.2 Gerenciamento de interesse baseado em classe

No gerenciamento de interesse baseado em classe cada entidade do sistema, objetos e usuários, são classificadas. Um usuário antes de se conectar no sistema registra quais classes ele tem interesse de receber mensagens e receberá apenas mensagens das entidades nas quais está registrado.

Por exemplo, em simulações militares, a infantaria estão agrupadas em um pelotão e eles não estão interessados nas comunicações entre as esquadras de vôo no céu (LEE et al., 2007). Isso permite que uma filtragem mais detalhada de mensagens do que o gerenciamento de interesse baseado em proximidade (LEE et al., 2007).

2.2.3 Gerenciamento de interesse hibrida

Esse método de gerenciamento combina os mecanismos de filtragem do gerenciamento baseado em proximidade com o do gerenciamento baseado em classe.

Cada grupo *multicast* pode ser associado a um conjunto de tipos e localizações de entidades. Estas projeções com múltiplos atributos se restringem aos valores de uma ou mais variáveis nos pacotes das entidades, que deve

se encontrar entre um intervalo específico ou ser igual a um valor específico (RODRIGUES et al., s.d.).

Por exemplo, um filtro de mensagens pode formar um grupo *multicast* que contem objetos da classe pessoas que estejam localizados na região limitada pelas coordenadas (10, 20) e (50, 100).

2.3 Controle de concorrência

Desde que um AVD seja altamente interativo e dinâmico, objetos e seus atributos devem ser modificados frequentemente. Objetos podem ser criados ou destruídos dinamicamente (ERASLAN et al., 2007).

Quando um atributo de qualquer objeto é atualizado, é enviada uma mensagem de atualização para notificar os demais usuários sobre a atualização e manter uma visão consistente do ambiente para todos os participantes. O problema é quando mais de um usuário tenta manipular o mesmo objeto ao mesmo tempo.

A arquitetura do sistema deve suportar o controle de concorrência para gerenciar ações conflitantes dos usuários. Se um usuário tenta abrir uma porta enquanto outro está tentando fechá-la, o que acontece? (ERASLAN et al., 2007)

Um mecanismo de relação de posse é necessário para permitir que somente um usuário possa manipular um objeto por vez. Existem três abordagem para o controle de concorrência (LEE et al., 2007), abordagem pessimista, abordagem otimista e esquema de predição.

Na abordagem pessimista o objeto é bloqueado até que uma requisição de posse feita explicitamente do usuário para o objeto seja aceita, permitindo que o usuário manipule o objeto. A vantagem dessa abordagem é a simplicidade de implementação. Por outro lado se o atraso de comunicação for elevado pode diminuir a interatividade do ambiente, já que um objeto só pode ser manipulado se o requisitante receber permissão.

A abordagem otimista permite que objetos sejam manipulados pelos usuários sem que seja necessária a requisição de posse. Quando um conflito acontece, um reparo deve ser feito. Isto não somente torna o sistema complexo, mas também os usuários perplexos com o refazer e desfazer de ações previas na interface do usuário (LEE et al., 2007).

O controle de concorrência baseado na predição fica entre a abordagem pessimista e otimista. As chaves do controle de concorrência baseado na predição são: a precisão de escolher uma requisição de posse de um conjunto

de muitas requisições e atendê-la em tempo-real, e passá-la para o próximo requisitante imediatamente quando a atual predição se torna incorreta (LEE et al., 2007).

No entanto, no esquema atual, todas as requisição de propriedade são difundidas para todos os usuários inclusive o atual proprietário. Como o AVD aumenta de tamanho, o número de mensagens recebidas em cada potencial proprietário também aumenta, assim sobrecarregando a rede (LEE et al., 2007).

2.4 Replicação de dados

Para aumentar a interatividade normalmente se replica os dados do ambiente virtual nas estações clientes. Com isso também se diminui o fluxo de dados inicial quando um cliente se conecta ao ambiente.

Cada cliente atualiza seus dados replicados sempre que fizer uma alteração local ou receber uma notificação de alteração de outros clientes.

Algumas abordagens utilizam replicação de dados parcial, onde somente os dados de objetos no qual o usuário necessite são copiados no computador do usuário.

Os dados dos objetos são armazenados em um servidor, quando uma estação cliente acessa pela primeira vez uma área de interesse específica, as informações sobre os objetos são armazenadas em uma base de dados local.

Estas estações cliente têm as mesmas propriedades do modelo de base de dados distribuída, porém com uma base de dados local capaz de armazenar e gerenciar as informações de estado dos objetos de uma determinada área do ambiente virtual, distribuindo assim a carga de processamento da estação servidora e garantindo a consistência das demais bases de dados (CORREIA, 2005).

2.4.1 Distribuição de carga

Particionar um mundo virtual em múltiplas regiões e distribuir as responsabilidade pela gestão das regiões em vários servidores pode significativamente reduzir a carga de trabalho de cada servidor, o que permite um sistema que suporte um número maior de usuários simultâneos no ambiente virtual (LEE et al., 2007).

No entanto uma má distribuição de processamento entre os servidores pode gerar servidores sobrecarregados. Usuários conectados a servidores sobrecarregados teriam uma experiência de interatividade prejudicada devido ao atraso

de processamento provocado pelo excesso de usuários conectados no servidor.

Os atuais métodos de distribuição dinâmica de carga são classificados em três tipos:

1. **Distribuição local** o servidor sobrecarregado distribui parte do seu trabalho entre servidores que gerenciam áreas vizinhas a sua. A vantagem é que o servidor só necessita conhecer os dados relativos aos seus vizinhos e a migração de usuários é pequena.

2. **Distribuição global** o servidor distribui o trabalho extra entre todos os servidores do ambiente. Isso permite que mesmo em casos de estrema sobrecarga, o servidor consiga distribuí-la, o que não é possível na distribuição local.

3. **Um coordenador central** reparte todo o ambiente virtual utilizando um algoritmo de particionamento. Contudo reparticionar sobrecarrega e requer uma grande migração de usuários dependendo to tamanho do ambiente virtual e do número de servidores no sistema (LEE et al., 2007).

Na abordagem adaptativa o servidor sobrecarregado distribui a carga de trabalho entre um conjunto de servidores - além dos servidores vizinhos - de acordo com o *status* de trabalho dos servidores (LEE et al., 2007).

CAPÍTULO 3

O Framework AnyAVDNet

O *framework* AnyAVDNet foi desenvolvido utilizando a linguagem Java. A linguagem Java pode ser definida como simples, orientada a objeto, suporte a desenvolvimento *multithread*, com arquitetura neutra, portável, robusta e segura (ORACLE, 2011). Por essas características foi escolhida como plataforma de desenvolvimento.

Também foi utilizado a API SIGAR (*System Information Gatherer And Reporter*) no desenvolvimento do *framework* em conjunto com a biblioteca Jpcap. A API SIGAR segundo a definição em (HYPERIC, 2011) é uma API portável para coleta de informações do sistema, tais como:

- Memória do sistema, *swap*, cpu, carga média, o tempo de atividade.
- Detecção do sistema de arquivos.
- Detecção de interfaces de rede, informações de configuração, etc.

Essas informações estão disponíveis em vários sistemas operacionais, mas cada SO tem sua maneira de fornece-las. O SIGAR disponibiliza uma API para acessar essas informações independentemente da plataforma. SIGAR tem implementações em diversas linguagens, dentre elas Java, C#, Perl e Ruby. O SIGAR está sob a licença Apache License V2.0.

O Jpcap (FUJII, 2007) é uma biblioteca Java para captura e envio de pacotes de rede permitindo manipulação de dados em baixo nível. Pode capturar pacotes IPv4, IPv6, ARP/RARP, TCP, UDP e ICMPv4. Jpcap é um projeto *open source* e licenciado sobre a GNU LGPL.

3.1 A Arquitetura do AnyAVDNet

O AnyAVDNet foi construído com o intuito de fornecer à arquitetura AVDNet proposto por (CORREIA, 2005) um balanceamento de cargar entre os servidores primários de região utilizando o protocolo de comunicação *anycast* em nível de aplicação.

Na AVDNet, cada região do ambiente virtual é controlada por uma estação, definida como servidor de região, adotando o conceito de múltiplos servidores. Os clientes do ambiente se comunicam com os diferentes servidores enquanto se movimenta por este.

Estes servidores recebem atualizações de clientes e enviam as atualizações aos interessados desta informação, mantendo o controle dos clientes por ele atendido. Quando o participante é o primeiro a acessar uma determinada região, ele é identificado como servidor desta região, e todos os dados referentes aos objetos tridimensionais da região são recuperados na base de dados do servidor e são instanciados produzindo referências e tornando-os acessíveis, passando a ter a capacidade de processar requisições de outros clientes. Quando um novo participante entra no ambiente e sua região inicial se encontra ativa, deverá identificar o servidor e então solicitar os objetos lá contidos e suas referências.

O *framework* AnyAVDNet é apresentado como um complemento da camada de comunicação do AVDNet e propõe a criação de mais de uma estação para controlar cada região do ambiente, além de fornecer mecanismos para facilitar a tarefa dos clientes do AVDNet de encontrar, dentre estas estações controladoras, a melhor estação para estabelecer uma conexão, assim melhorando a distribuição da carga de processamento entre as estações servidoras e diminuindo a latência das trocas de mensagens, visto que o *anycast* entrega as mensagens para o servidor mais adequado, de acordo com a métrica *anycast*.

Existem duas abordagem para se trabalhar com *anycast*. A primeira é o *anycast* na camada de rede, também conhecido como IP *anycast*, onde a escolha do servidor é feita através de protocolos de roteamento que fazem uso de métricas disponpiveis na camada de rede, como número de saltos e tempo de resposta do servidor.

Segundo (MA; ZHOU; ZHANG, 2009), o principal obstáculo dessa abordagem é o fato de o protocolo utilizado na escolha do servidor, não tenha sido incluído nas especificações do *anycast*. Problemas de escalabilidade nas tabelas de roteamento e no esquema de seleção ainda são questões em aberto no IP *anycast*.

O IP *Anycast* não funciona bem com grupos altamente dinâmico e sua

implantação tem sido prejudicado por preocupações com segurança, escalabilidade, faturamento e com o número de grupos. (CASTRO et al., s.d.)

A segunda abordagem *anycast*, é o *anycast* em nível de aplicação. Nessa abordagem o problema da escolha de um servidor é resolvido na camada de aplicação, sem o envolvimento dos roteadores. A escolha do servidor tem como base alguns indicadores de desempenho, tais como a capacidade do servidor, tempo de resposta, a carga do servidor, etc. (WU; HWANG; HO, s.d.)

Por ser mais flexível, o *anycast* em nível de aplicação foi a abordagem escolhida para implementar o *framework* AnyAVDNet.

A estratégia abordada no AnyAVDNet é ilustrada na Figura 3.3. O servidor *Anycast*, ao receber uma conexão de um cliente, verifica se existem servidores primários para a região na qual o cliente deseja se conectar. Se não existirem servidores, esse cliente passa a ser o servidor primário desta região. Caso contrário, o servidor *Anycast* envia ao cliente uma lista com os endereços de todos os servidores para aquela região.

Ao receber a lista, o cliente envia para cada servidor na lista, uma solicitação requisitando o valor de sua métrica *anycast*. Cada servidor, então calcula sua métrica correspondente e envia para o cliente.

De posse das métricas *anycast* dos servidores, o cliente tem condições de escolher o servidor mais apto para realizar a conexão.

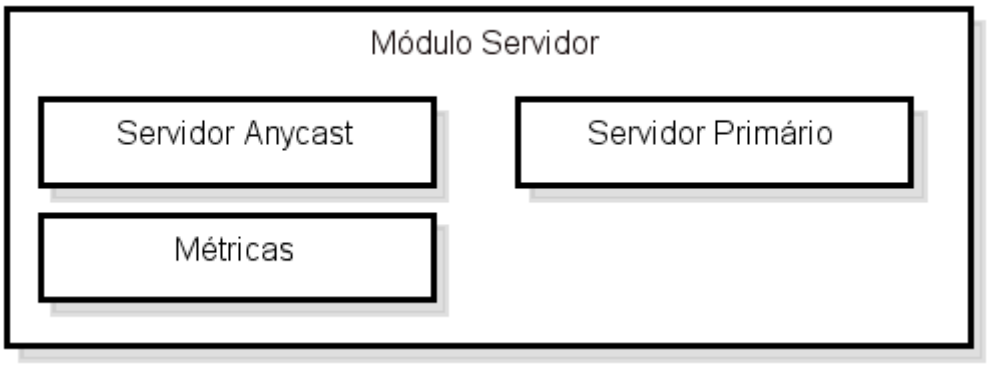

Figura 3.1: Módulo Servidor

Figura 3.2: Módulo Cliente

A arquitetura do AnyAVDNet é dividida em dois módulos, o módulo Cliente

e o módulo Servidor. O módulo Servidor disponibiliza funcionalidades para a criação de um servidor *anycast* e adiciona aos servidores de região a capacidade de recolher dados para o cálculo de sua métrica *anycast* e enviá-las ao cliente. Um servidor *anycast*, na arquitetura proposta, tem a responsbilidade de manter o endereço de todos os servidores primários de cada região, para que um cliente, ao se conectar possa escolher um servidor de região que esteja em melhor condição de atendê-lo.

O módulo Cliente apresenta funcionalidades que permitem ao cliente do AVD conectar-se ao servidor *anycast* e também solicitar aos servidores primários de região suas respectivas métricas *anycast*.

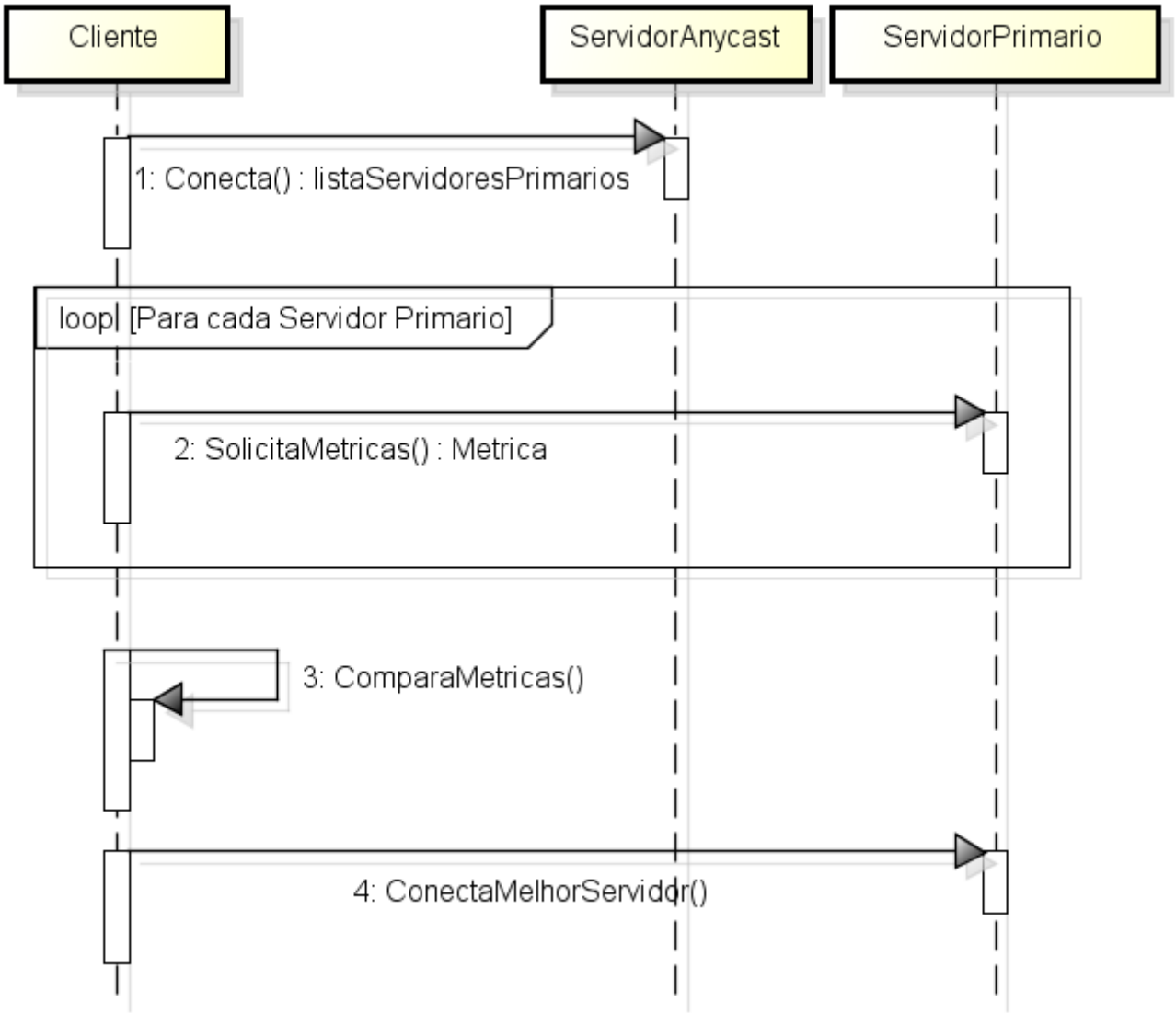

Figura 3.3: Sequência de funcionamento do AnyAVDNet

3.2 Implementação do AnyAVDNet

O *framework* AnyAVDNet foi escrito utilizando a linguagem Java e suas classes são apresentadas na Figura 3.4.

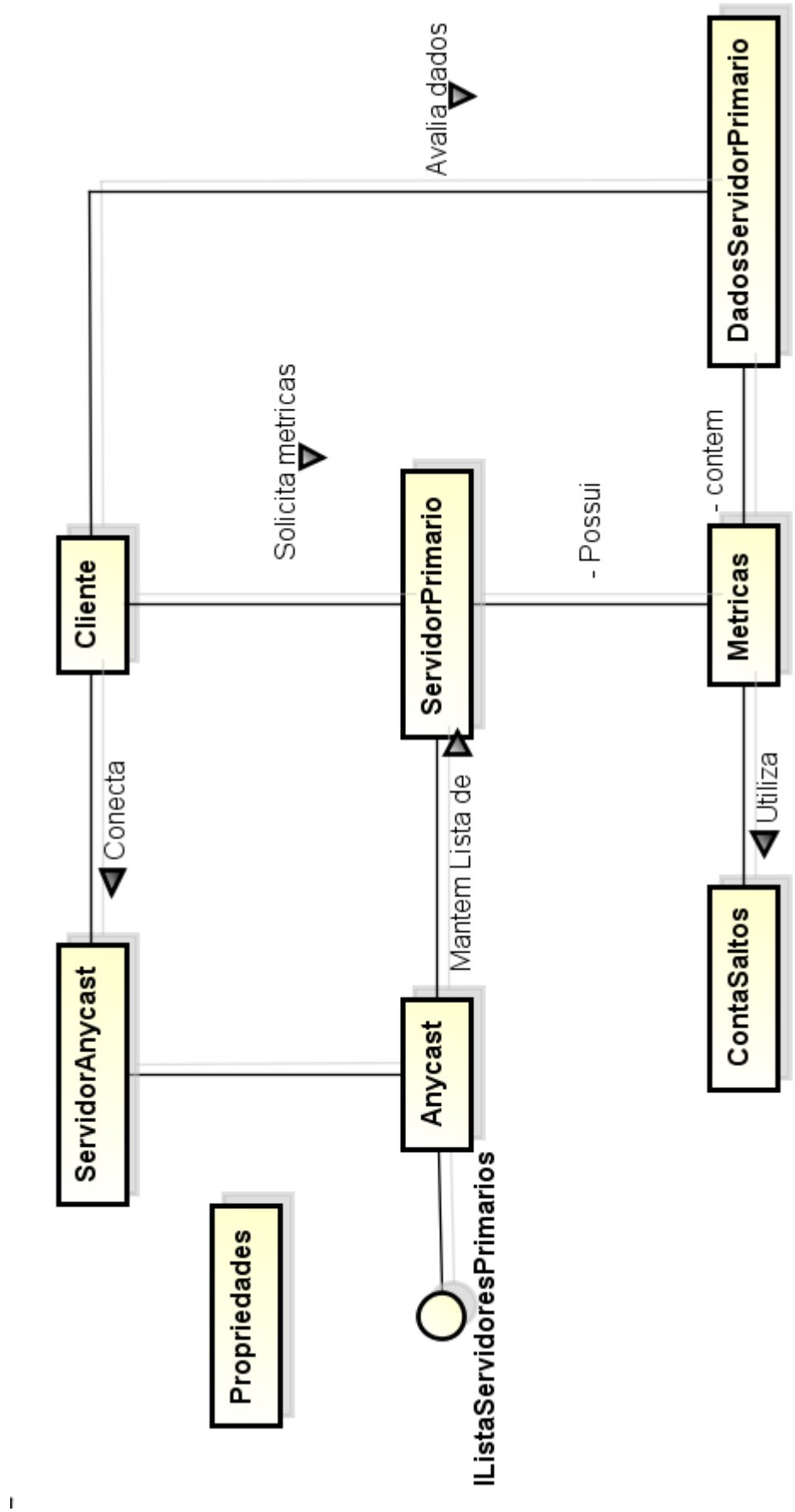

Figura 3.4: Diagrama de classes do AnyAVDNet

Na classe *ServidorAnycast*, apresentada na Figura 3.5, implementa da *interface Runnable* e sobrescreve o método *public void run(void)* para aguardar conexões dos cliente. Quando uma conexão é recebida por meio de um *socket* uma instância da classe *Anycast* é criada para tratar essa conexão e o Servidor Anycast volta a aguardar outras conexões.

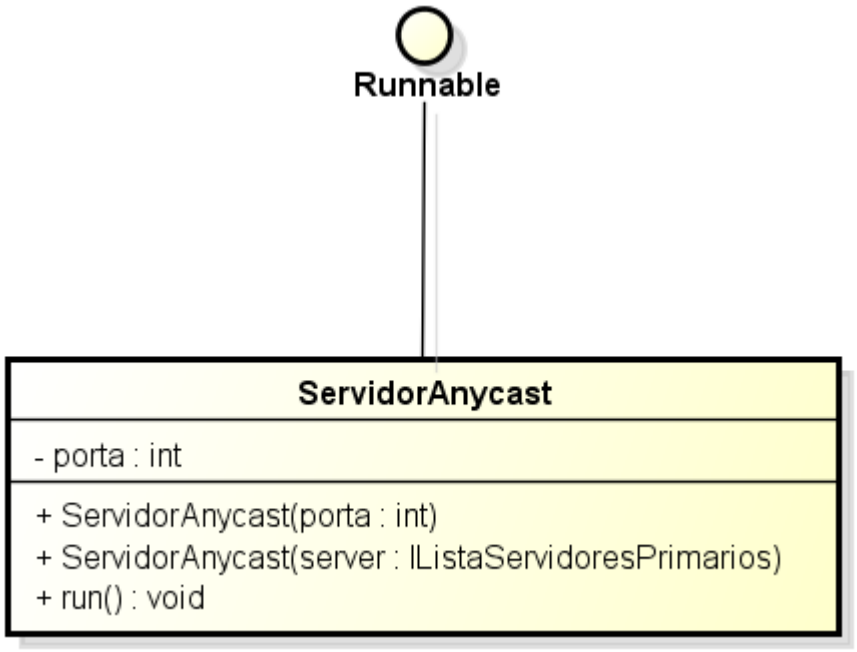

Figura 3.5: Classe ServidorAnycast

A classe *Anycast*, apresentada na Figura 3.6, também implementa a *interface Runnable* e possui o método *processaConexao*, responsável em tratar a conexão do cliente verificando a existência de servidores primários de região. Se não houverem servidores primários, a classe Anycast envia uma mensagem para o cliente se tornar o servidor primário desta região, caso contrário é enviado para o cliente uma lista com endereços desses servidores.

Para tornar a manipulação da lista de servidores primários mais flexível, a classe *Anycast* possui um atributo *servidoresPrimarios* do tipo *IListaServidoresPrimario*, ilustrada na Figura 3.7. *IListaServidoresPrimario* é uma *interface* que declara métodos comuns para a manipulação da lista de servidores primários, deixando a cargo da aplicação usuária do AnyAVDNet implementar esses métodos da forma que melhor atenda sua necessidade, sem impacto na funcionalidade do *framework*.

O usuário do *framework* deve implementar essa *interface* permitindo ao *framework* manipular a lista de servidores sem a preocupação de como essa lista é organizada.

A classe *ServidorPrimario*, apresentada na Figura 3.8 é instânciada no cliente quando este se torna um servidor de região. Tem a finalidade de coletar as informações necessárias para o cálculo da métrica, através do atributo

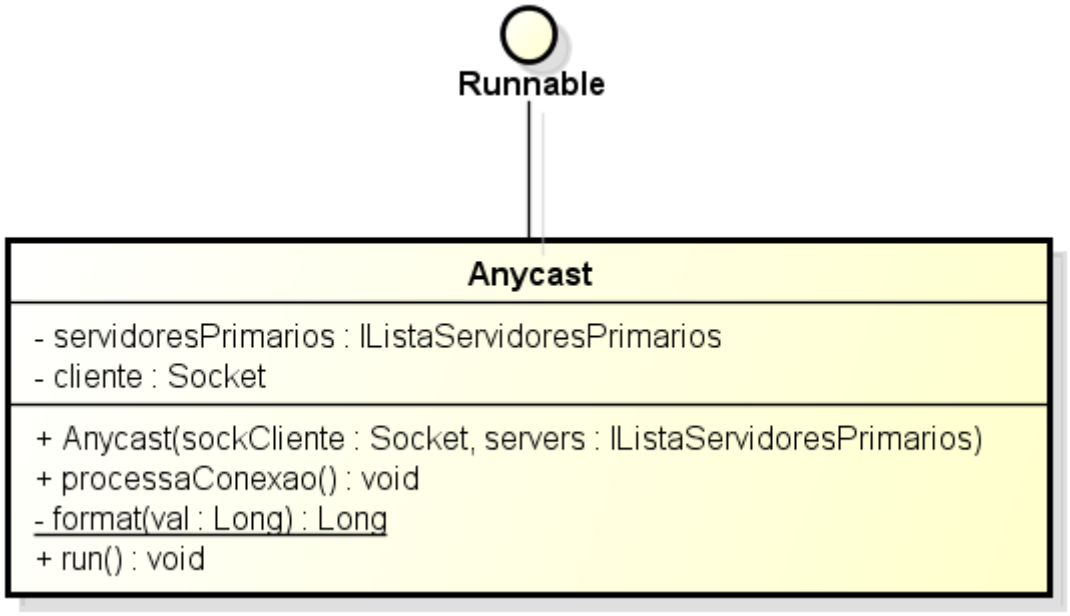

Figura 3.6: Classe Anycast

Figura 3.7: Interface IListaServidorPrimario

metrica, e esperar conexões de outros clientes para enviar essas métrica ao solicitante.

A classe *Cliente*, apresentada na Figura 3.9, se conecta a um servidor *anycast*, através do método *obtemMelhorServidor()*, solicitando os endereços dos servidores de região disponíveis. Ao receber esses endereços, é solicitado aos servidores primários de região suas respectivas métricas e baseado nessas métricas escolhe o servidor para se conectar.

As duas principais características da classe Cliente são o atributo metricaMinima, do tipo double, que armazena o valor mínimo que a métrica do servidor primário deve possuir para que o cliente possa se conectar, caso a métrica *anycast* do servidor primário seja inferior ao valor de metricaMinima, o cliente passa a ser também um servidor primário.

A segunda principal característica da classe *Cliente* é o método *obtemMe-*

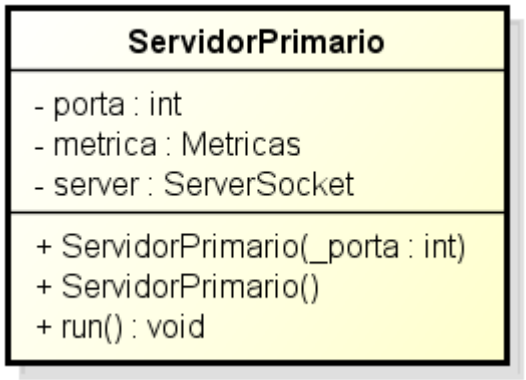

Figura 3.8: Classe ServidorPrimario

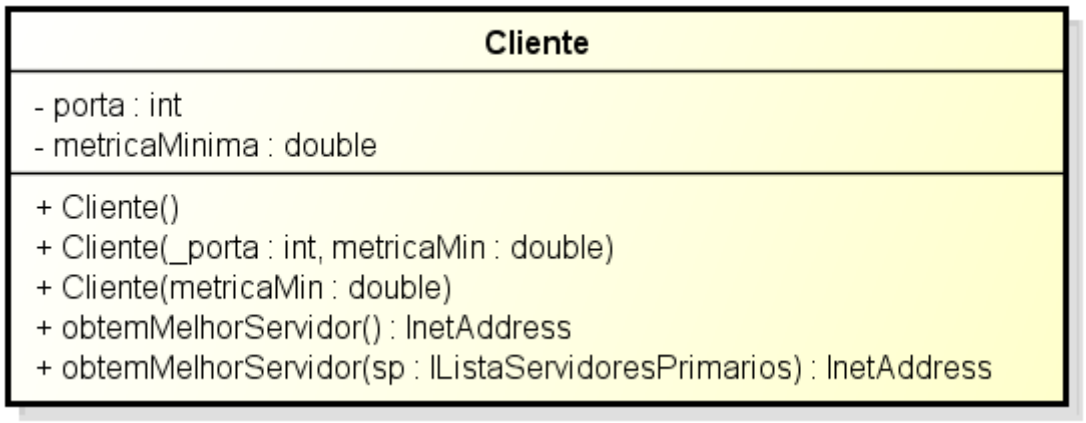

Figura 3.9: Classe Cliente

lhorServidor. Como mostrado na Figura 3.9, existe uma sobrecarga desse método, tanto na versão sem parâmetros como na versão que recebe uma lista de endereços de servidores primários, ambas retornam um objeto do tipo InetAddress, que contém o endereço, IPv4 ou IPv6, do servidor escolhido, caso não exista servidores primários, ou nenhuma das métricas dos servidores primários estejam acima do valor de *metricaMinima*, os métodos retorna **null**.

A classe *Metrica*, apresentada na Figura 3.10, principal classe do *framework*, é responsável pela coleta dos dados do servidor primário de região, tais como tempo de CPU disponível, quantidade de memória RAM livre e o número de saltos IP entre o cliente e o servidor primário.

Para recolher as informações de quantidade de memória, tempo de CPU foi utilizado a API SIGAR (HYPERIC, 2011).

Para obter a quantidade de saltos IP é necessário a manipulação das informações dos cabeçalhos dos pacotes. A API padrão do Java fornece somente abstrações do TCP (*Socket*) e do UDP (*Datagram*). Essas abstrações não permitem a manipulação destas informações.

Para isso foi criada a classe ContaSaltos, Figura 3.12, que implementa a funcionalidade de contar os saltos IP do servidor de região até o cliente utili-

zando as classes disponíveis na biblioteca Jpcap (FUJII, 2007), que possuem métodos para manipulação, visualização e analizes dos dados dos pacotes através do Java.

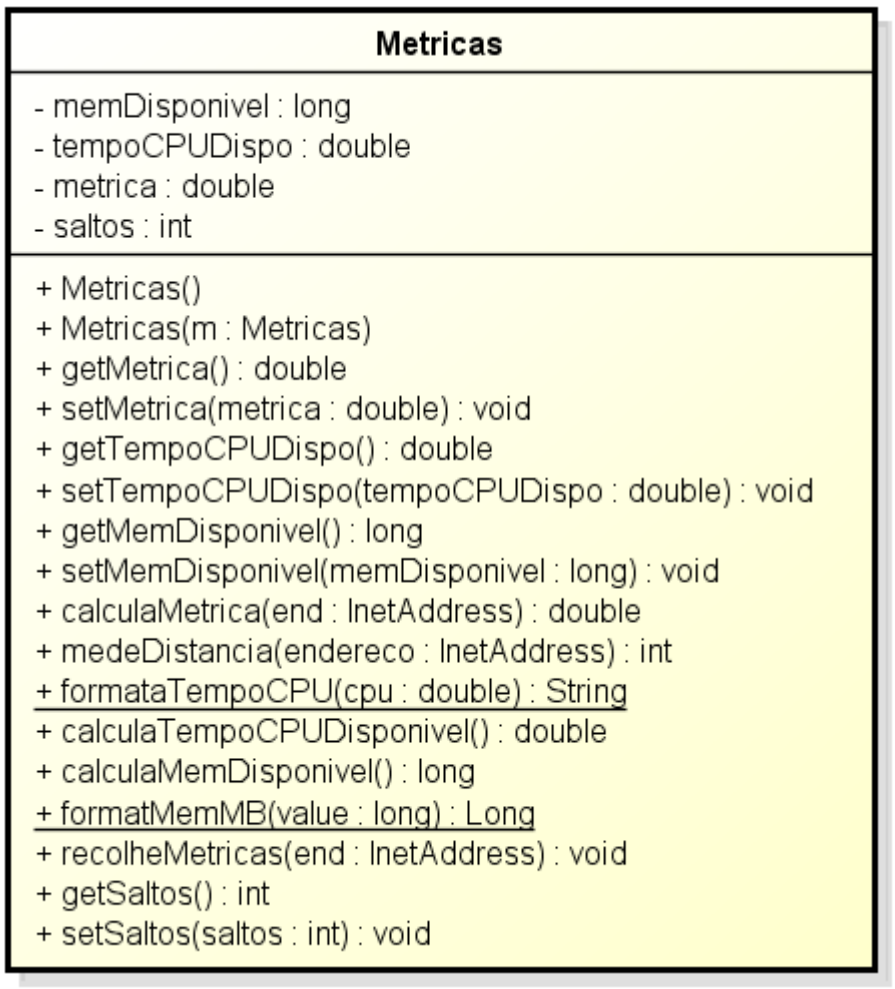

Figura 3.10: Classe Metricas

Através do método *calculaMetrica* a classe *Metricas* retorna o valor da métrica do servidor primário, realizando uma média ponderada com os valores de saltos IP, quantidade de memória RAM disponível e tempo de CPU livre.

A classe *DadosServidorPrimario*, Figura 3.11, serve para manter a métrica *anycast* do servidor e o seu endereço, e possui somente métodos *getter* e *setter*.

Para tornar o *framework* flexível, foi criado o pacote *utils*. Nesse pacote encontra-se a classe *Propriedades*, que tem a finalidade de ler um arquivo externo de configuração e carregar seu conteúdo ao *framework* AnyAVDNet.

Arquivos de configurações são arquivos texto com a estrutura <chave>=<valor> e são usados para ajustar atributos na aplicação Java e são carregados uma unica vez na aplicação.

A classe *Propriedades* procura pelo arquivo propriedades.*properties* na pasta onde o *framework* está rodando, caso esse arquivo não exista, valores padrões são ajustados.

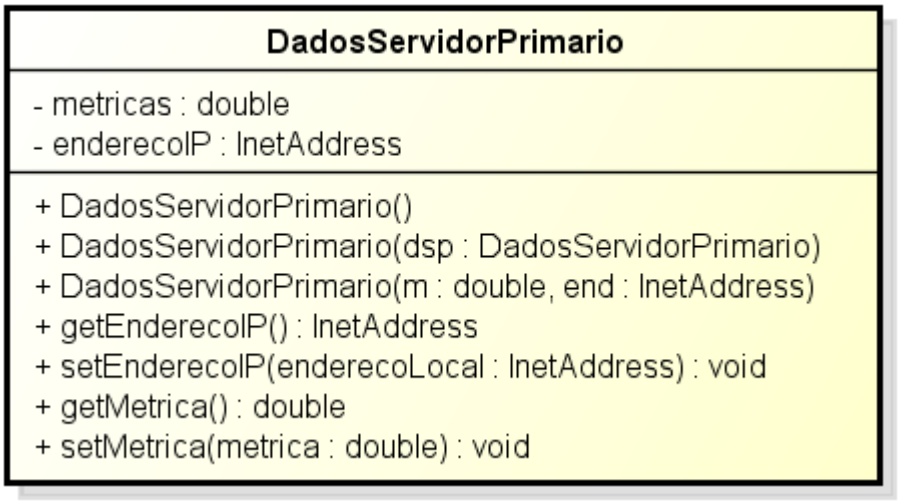

Figura 3.11: Classe DadosServidorPrimario

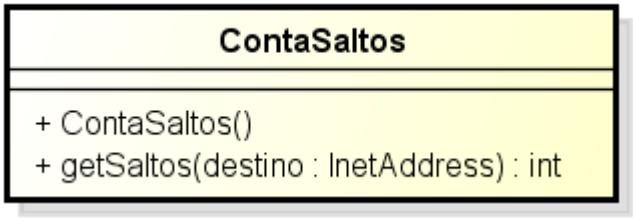

Figura 3.12: Classe ContaSaltos

As chaves disponíveis no propriedades.*properties* são:

enderecoServidorAnycast: endereço IP do servidor Anycast.

portaServidorAnycast: porta na qual o servidor anycast espera por conexões.

portaServidorPrimario: porta na qual o servidor primário de região espera por conexões.

portaCliente: porta na qual o cliente espera por conxões.

pesoMemoria: indica o peso da quantidade de memória no calculo da métrica *anycast*.

pesoCPU: indica o peso do tempo de CPU livre no calculo da métrica *anycast*.

pesoSalto: indica o peso do número de saltos no calculo da métrica *anycast*.

metricaMinima: valor mínimo da métrica *anycast* de um servidor primário para aceitar conexões de clientes, deve ser em notação (ciêntifica) Exponencial (Ex.: 5.96e9).

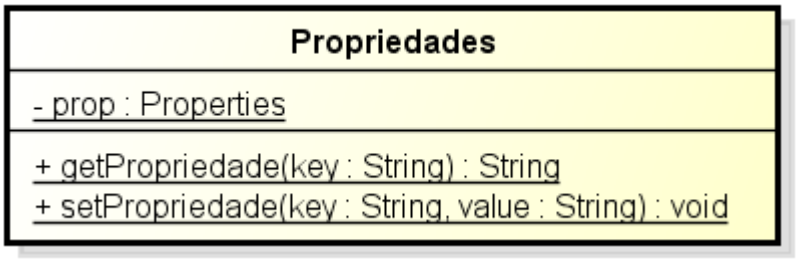

Figura 3.13: Classe Propriedades

3.3 Testes e Resultados

Para avaliar as funcionalidades do *framework* AnyAVDNet, foi criado uma aplicação para simular as açoes de Clientes, servidor Anycast e servidores Primários.

A *interface* do Servidor *Anycast*, apresentada na Figura 3.14 cria um objeto do tipo *ServidorAnycast*, Figura 3.5 e fica a espera de conexões de clientes. A interface IListaServidoresPrimarios foi implementada pela classe ListaServidoresPrimarios e simplesmente mantêm um atributo do tipo LinkedList<InetAddress>, para manter a lista de servidores primários e um JTextArea usado na Tela do Servidor Anycast para mostrar os servidores primários existentes. Para poder manipular as informações na tela adquadamente, se fez necessário a criação da classe TesteAnycast, que é uma extensão da classe Anycast com a funcionalidade de atualizar a exibição de servidores primários conectados no servidor Anycast.

Para a criação da *interface* Cliente foram criadas as classes TesteCliente, que extende da classe Cliente, Figura 3.9 e adiciona atributos para exibição das informações sobre o cliente na Tela Cliente. Ao buscar o servidor primário para se conectar, se não existir um servidor primário adquado o cliente passar a ser um servidor primário, a tela de Cliente instância a Tela de Servidor Primário, ilustrada na Figura 3.15.

A *interface* de Servidor Primário exibe o IP local do servidor primário, e também exibe informações como mémoria RAM disponível e porcentagem de CPU livre atualizadas a cada 5 segundos.

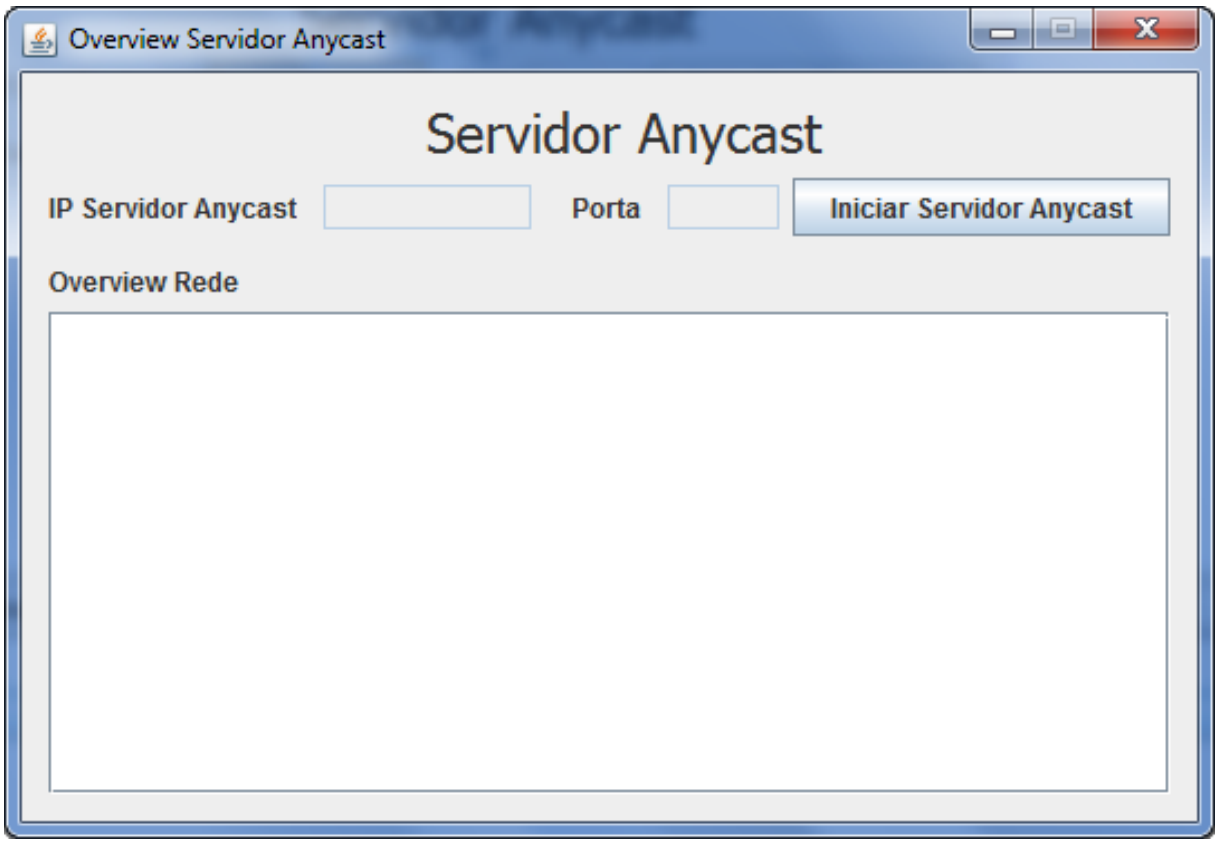

Figura 3.14: Interface do Servidor Anycast

Figura 3.15: Interface do Servidor Primario

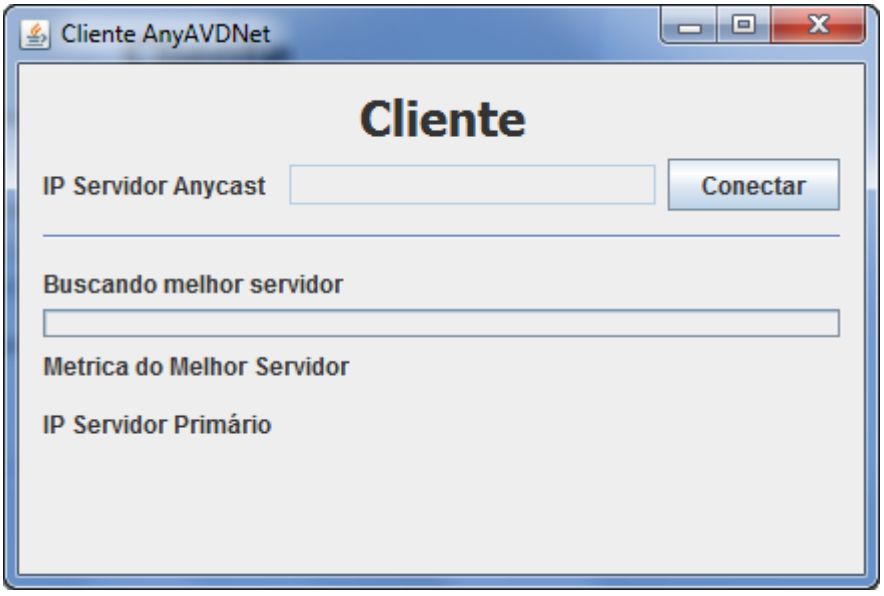

Figura 3.16: Interface do Cliente

3.3.1 Testes Realizados

Foram utilizados três computadores para a realização da simulação. Os computadores P1 e P2 com processador Intel Core 2 Duo de 2 GHz e 2GB de memória RAM e o computador P3 com processador Intel Celeron de 2,8GHz e 2,5 GB de memória RAM.

O arquivo propriedades.*properties* nos três ambientes tem as seguintes configurações:

```
#Arquivos de propriedades do framework
#criado por Ricardo Oliveira
enderecoServidorAnycast=192.168.0.192
portaServidorAnycast=39876
portaServidorPrimario=39877
portaCliente=39878
pesoMemoria=0.5
pesoCPU=0.4
pesoSalto=0.1
metricaMinima=1.0
```

Em um primeiro teste foi realizado os seguintes passos:

1. O computador P1 foi escolhido como Servidor Anycast, com IP 192.168.0.192.
2. O computador P2 foi o primeiro a se conectar no servidor Anycast e se tornou um servidor primário com IP 192.168.0.134.
3. O computador P3 se conectou ao servidor Anycast e obteve o IP do servidor primário (P2) e solicitou sua métrica, e obteve como resultado

   ```
   Metrica = 6.817239045E8
   Endereço Servidor = /192.168.0.134
   ```

   como no arquivo de configuração o valor da *metricaMinima* estava definido como 1.0 o computador P3 se tornou cliente de P2, como esperado.

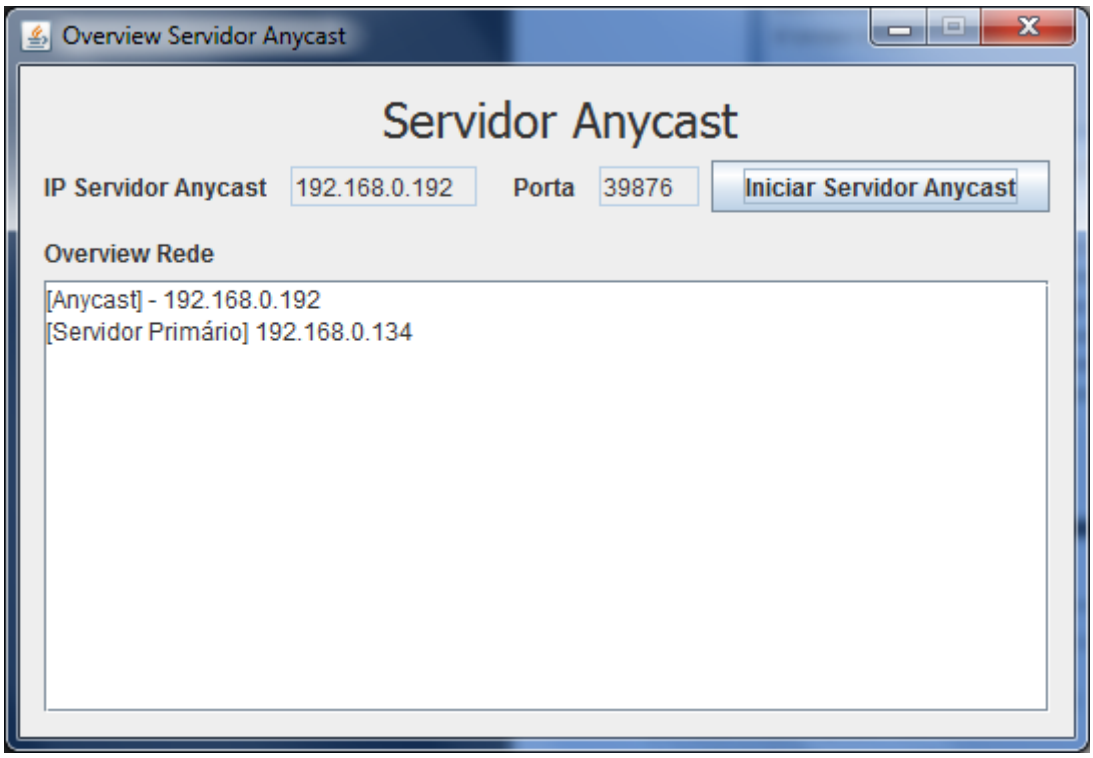

Figura 3.17: Interface do Servidor Anycast - Caso de Teste 1

Figura 3.18: Interface do Servidor Primario - Caso de Teste 1

Figura 3.19: Interface do Cliente - Caso de Teste 1

No segundo teste realizado, o computador P1 continua sendo o servidor

Anycast. E para forçar a criação de dois servidores primários, o valor do atributo metricaMinima no arquivo de configuração no computador P3 foi ajustado para 1.0e9 (1.000.000.000).

Os passos realizados neste segundo teste foram:

1. O computador P1 foi escolhido como servidor Anycast, IP 192.168.0.192.
2. O computador P2 foi escolhido para realizar a primeira conexão com o servidor *Anycast*, assim se tornando um servidor primário.
3. O computador P3 foi o segundo a se conectar ao servidor *Anycast* e obteve o endereço de P2 (Servidor Primário)
4. P3 se conecta a P2 e solicita sua métrica *anycast* obtendo o resultado

   ```
   Metrica = 5.813862403829489E8
   Endereço Servidor = /192.168.0.134
   ```

   Como 5.813862403829489E8 (581.386.240,3829489) é menor que o valor da metricaMinima 1.0e9, P3 também se torna um servidor primário.
5. Em P1 também é instaciado a aplicação cliente, que se conecta ao servidor *Anycast* e obtem os endereços dos servidores primários P2 e P3.
6. P1 solicita as métricas *anycast* de P2 obtendo

   ```
   Metrica = 5.782364164156157E8
   Endereço Servidor = /192.168.0.134
   ```

   e de P3

   ```
   Metrica = 1.0960322562064E9
   Endereço Servidor = /192.168.0.195
   ```

   Como a métrica *anycast* de P3 1.0960322562064e9 (1.096.032.256,2064) é maior que a de P2 5.782364164156157e8 (578.236.416,4156157) P1 passa a ser cliente de P3

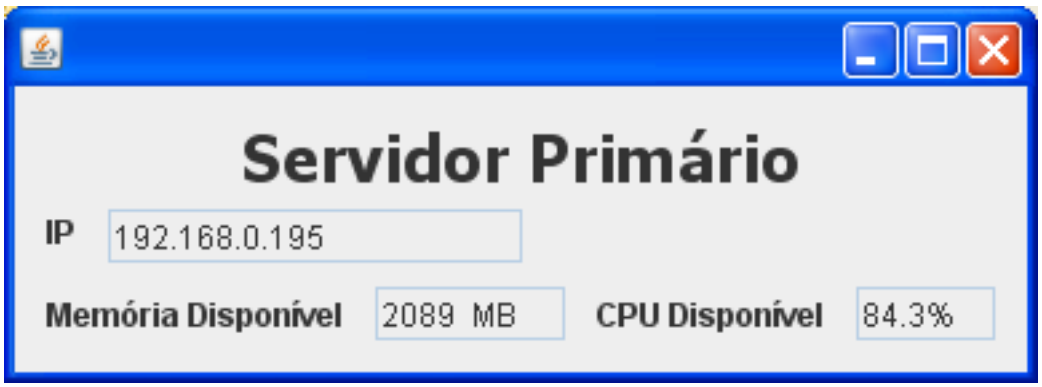

Figura 3.22: Interface do Servidor Primário P3 - Caso de Teste 2

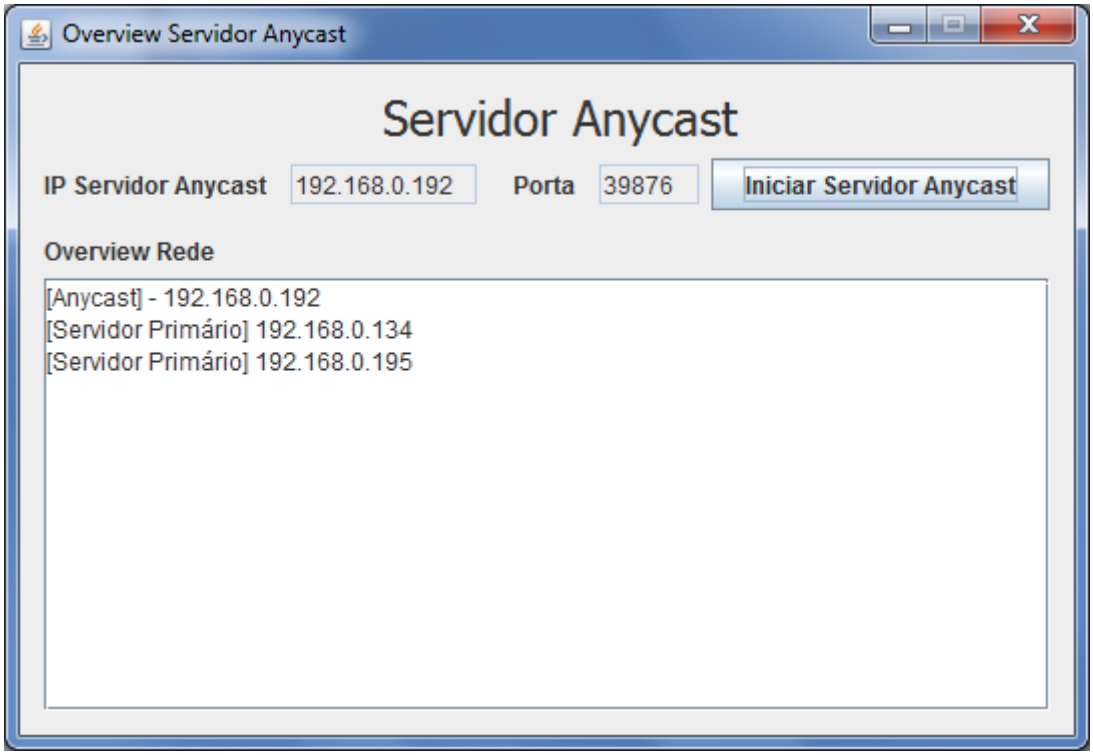

Figura 3.20: Interface do Servidor Anycast - Caso de Teste 2

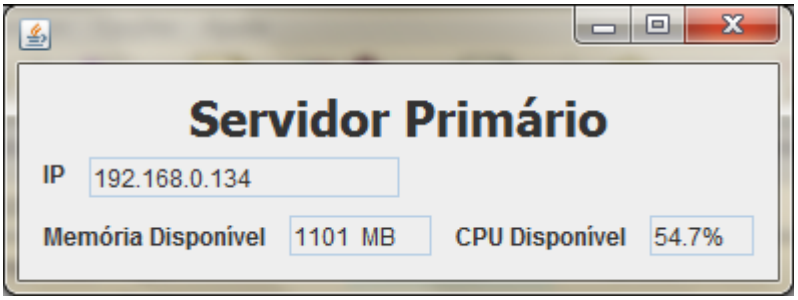

Figura 3.21: Interface do Servidor Primário P2- Caso de Teste 2

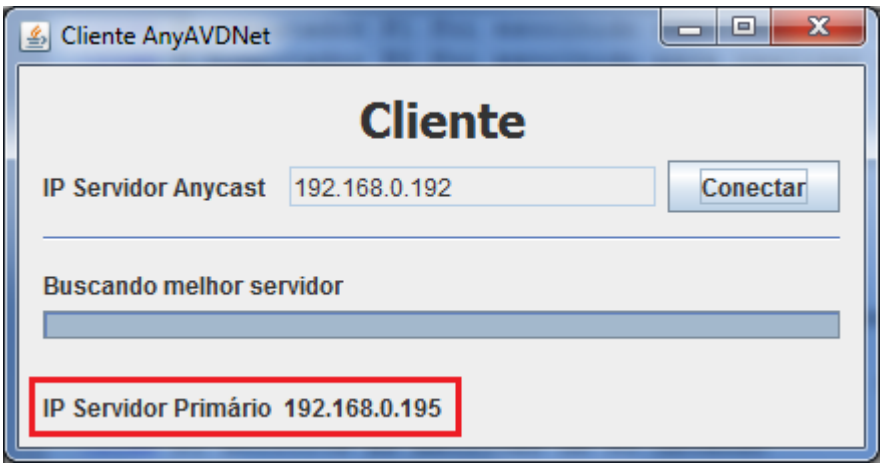

Figura 3.23: Interface do Cliente - Caso de Teste 2

Com a análise desses resultados, fica evidente que o *framework* AnyAVDNet é capaz de escolher um servidor primário dentre um conjunto de servidores, baseando-se em suas métricas *anycast*, proporcionando um balanceamento de carga entre esses servidores e aumentando a escalabilidade do sistema.

CAPÍTULO 4

Conclusão

4.1 Conclusão

Como mostrado ao longo desta monografia a concepção de Ambientes Virtuais Distribuidos envolve conceitos de muitas áreas distinta, como redes de computadores, computação gráfica, interface homem máquina e banco de dados.

As maiores limitações para o desenvolvimento de AVDs de grande escala são impostas pelo sistema de comunicação. Conforme o número de usuários cresce, aumenta a quantidade de banda necessária para a troca de informação em tempo real entre as estações para atualizar o ambiente e mantê-lo em um estado consistente para todos os usuários conectados.

Reduzindo a carga dos servidores pode-se diminuir as latências das transmissões causadas pelos congestionamentos e consequentemente, as atualizações de mudança de estado do ambiente serão enviadas mais rapidamente.

Tendo isso em mente, esta trabalho teve como objetivo apresentar uma forma de balanceamento de carga dinâmico entre os servidores primários da arquitetura do AVDNet, desenvolvendo um *framework*, o AnyAVDNet, que utilize as vantagens do protocolo *anycast* em nível de aplicação para balanceamento de carga entre os servidores primários de região e diminuir a latência da troca de pacotes na rede pelos participantes do AVD.

O AnyAVDNet é um *framework* portável e flexível. E por essas características o AnyAVDNet pode ser utilizado facilmente na construção de outros sistemas distribuídos ou agregar a funcionalidade de distribuição de carga em

sistemas já existentes.

4.2 Trabalhos Futuros

O AnyAVDNet utiliza tempo de CPU livre, quantidade de memória RAM disponível e número de saltos IP entre o servidor primário e o cliente para o cálculo da métrica *anycast* dos servidores. Apesar da relevância dessas medidas para o melhor balanceamento de carga entre servidores outras medidas podem ser adicionadas no cálculo da métrica *anycast* de forma a tornar o balanceamento mais eficiente, como tempo de resposta e a quantidade de clientes já conectados no servidor.

O AnyAVDNet calcula a métrica *anycast* realizando uma média ponderada entre as informações recolhidas no servidor primário. Mas é necessário um estudo mais profundo sobre a relevância de cada componente no calculo desta média ponderada.

Referências Bibliográficas

CAPIN, T. K. et al. ***Avatars in Networked Virtual Environments***. [S.l.]: John Wiley and Sons, 1999.

CASTRO, M. et al. **Scalable application-level anycast for highly dynamic groups**. [S.l.], s.d.

CORREIA, R. C. M. **AVDNet-Arquitetura para Ambientes Virtuais Distribuídos Escaláveis Baseada na Infra-Estrutura Atual da Internet.** Tese (Ciência da Computação) — Instituto Tecnológico de Aeronáutica, São José dos Campos, SP., 2005.

DIEHL, S. **Distributed Virtual Worlds**: Foundations and implementation techniques using vrml, java, and corba. New York: Springer, 2001.

DUNDES, L. W. M. Graduação em Ciência da Computação, **AVDNet6 - Framework de gerenciamento de aplicações distribuídas baseadas no protocolo IPv6**. Presidente Prudente: [s.n.], 2008. 61 f.

ERASLAN, M. et al. **A scalable network architeture for distributed virtual environments with dynamic qos over ipv6**. Ontario Canadá, 2007.

FUJII, K. **A Java library for capturing and sending network packets**. 2007. Disponível em: <http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/>.

FUNKHOUSER, T. Ring: A client-server system for multi-user virtual environments. In: **Proceedings of the ACM Symposium on Interactive 3D Graphics**. New York: ACM Press, 1995. p. 85–92.

GREENHALGH, G.; BENFORD, S. Boundaries, awareness and interaction in collaborative virtual environments. In: **Proceedings of the Sixth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises**. [S.l.: s.n.], 1997. p. 193–198.

HYPERIC. **SIGAR, System Information Gatherer and Reported**. 2011. Disponível em: <http://support.hyperic.com/display/SIGAR/Home>.

JUNIOR, A. J. M. L. **ATAXIA: Uma arquitetura para viabilizaçcão de NVEś voltados para a Educação a Distância através da Internet**. Dissertação (Ciência da Computação) — Universidade Federal do Ceará, Fortaleza, 2000.

LEE, D. et al. **ATLAS A Scalable Network Framework for Distributed Virtual Environments**. Korea, 2007.

MA, Z.-Y.; ZHOU, J.; ZHANG, L. **A Scalable Framework for Global Application Anycast**. [S.l.], 2009.

ORACLE. **About the Java Technology**. 2011. Disponível em: <http:/-/download.oracle.com/javase/tutorial/getStarted/intro/definition.html>.

PARTRIDGE, C.; MENDEZ, T.; MILLIKEN, W. **RFC 1546 - Host Anycasting Service**. Novembro 1993.

RODRIGUES, L. C. R. et al. **Ambientes Virtuais Distribuídos**. [S.l.], s.d.

SINGHAL, S.; ZYDA, M. **Networked Virtual Environments**. [S.l.]: Addison-Wesley, 1999.

VANDEIR, E. Graduação em Ciência da Computação, **Protótipo de um ambiente virtual distribuído multiusuário**. Blumenau: [s.n.], 2001. 121 f.

WU, C.; HWANG, R.; HO, J. **A Scalable Overlay Framework for Internet Anycasting Service**. Taiwan, s.d.