Middleware QoSWare: Provendo Suporte à Qualidade de Serviço para Aplicações Avançadas

Suzana de F. Dantas, Arthur de C. Callado, Rodrigo S. B. G. Barbosa, Igor C. Cananéa, Paulo G. Barros, Djamel F. H. Sadok, Judith Kelner

Centro de Informática, Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851, 50732-970, Recife – PE, Brasil.

{sfd, acc2, rsbgb, icc, pgb, jamel, jk}@cin.ufpe.br

Abstract. *The QoSWare middleware was developed to provide network communication transparency for application developers and network Quality of Service (QoS) support for real time applications such as action games, multimedia and virtual reality applications. QoSWare also offers useful building blocks "services" that could be required by a real time application, such as movement prediction. This paper presents the QoSWare architecture and a case study designed to evaluate it based on a real time action game developed as part of this evaluation process. It was observed that movement prediction techniques could allow users to negotiate a less expensive QoS level with their Internet Service Provider without loss of playability.*

Resumo. *O middleware QoSWare foi desenvolvido para prover transparência de comunicação para os desenvolvedores de aplicações e suporte à Qualidade de Serviço (QoS) para aplicações de tempo-real tais como jogos de ação, vídeo e realidade virtual compartilhada. O QoSWare também oferece algumas funções de apoio, dentre elas as de predição de movimento que podem ser solicitadas pela aplicação se necessário. Esse artigo apresenta a arquitetura do QoSWare e um estudo de caso para sua avaliação. Um jogo foi especialmente desenvolvido para os testes. Observou-se que técnicas de predição de movimento podem permitir que usuários negociem níveis de QoS menos exigentes e mais baratos sem comprometer a qualidade da aplicação.*

1 Introdução

Prover suporte às aplicações avançadas na Internet ainda é uma boa fonte de discussão e de oferta de soluções na comunidade científica. Entende-se por aplicações avançadas aquelas que requerem condições especiais da rede para funcionar satisfatoriamente, tais como jogos de ação, realidade virtual compartilhada e videoconferência. Esse artigo descreve alguns resultados obtidos na especificação de um *middleware* chamado QoSWare que gerencia qualidade de serviço (QoS) para aplicações de tempo-real.

O QoSWare não pretende ser uma plataforma de desenvolvimento a sistemas distribuídos. O QoSWare pode ser visto como um *toolkit* para os desenvolvedores de aplicações. Para tanto, o QoSWare baseia-se em dois princípios: ser facilmente integrável com qualquer aplicação e evitar um consumo adicional de recursos de CPU e de memória, como também de banda, que poderiam comprometer o desempenho da aplicação.

Além das garantias de QoS, a principal motivação para o desenvolvimento do QoSWare foi a de promover uma abstração da rede para as aplicações em geral. Assim, a implementação da comunicação de dados fica a cargo do *middleware,* cabendo à aplicação apenas invocá-lo, utilizando a interface disponível para esse fim. Neste artigo, os autores procuram avaliar a eficiência do *middleware* e seu suporte a QoS.

Algumas ferramentas de apoio, tais como algoritmos de predição de movimento e de convergência, foram implementados no QoSWare para serem opcionalmente utilizados pela aplicação. Utilizando um algoritmo de predição bastante simples foi possível identificar situações onde a combinação desse recurso com um cenário de qualidade de serviço menos exigente (PHB[1] AF no DiffServ) passou a ser tão aceitável quanto um cenário ideal, sem perdas e/ou atrasos. Com isso, um usuário poderia negociar um serviço mais barato, sem comprometer a qualidade da aplicação.

Para avaliar o QoSWare, um jogo de ação multiusuários foi desenvolvido e está detalhado neste trabalho. Para os experimentos, em vez de simulações ou experimentações em ambientes reais, um *testbed*[2] foi implementado para emular o comportamento da Internet. Esse *testbed* envolve várias máquinas e softwares e está descrito mais adiante no Estudo de Caso.

As próximas seções estão divididas da seguinte maneira: a seção 2 discursa sobre alguns trabalhos relacionados e procura contextualizar a proposta do QoSWare; as terceira e quarta seções apresentam a arquitetura QoSWare e o Estudo de Caso, respectivamente; na quinta seção detalham-se os resultados obtidos e por fim, são traçadas as considerações finais a respeito do trabalho.

2 Trabalhos relacionados

Do ponto de vista do consórcio Internet2 [Internet2 2003], *middleware* é uma camada de software localizada entre a camada de rede e a de aplicação. As referências [Aiken 2000], [Bakken 2001], apresentam detalhes sobre características e funcionalidades de *middleware*. O projeto Da CaPO++ [Stiller et al 1999] descreve um *middleware* para desenvolvimento e suporte de/para serviços multimídia, em especial aplicações de vídeo. Todavia os principais focos desse projeto são requisitos de QoS para segurança e desenvolvimento de uma infra-estrutura para prover comunicação *multicast*.

Plataformas de desenvolvimento de aplicações distribuídas tais como CORBA e DCOM foram analisadas durante a fase de concepção do QoSWare, contudo são ambientes de desenvolvimento bem mais complexos, com objetivos bem maiores do que apenas prover suporte à comunicação com QoS. O dynamicTAO [Kon et al 2000] é um *middleware* em tempo-real que utiliza CORBA e permite a reconfiguração dinâmica de alguns parâmetros do sistema. Embora o dynamicTAO ofereça suporte a QoS, ele não o implementa. O QuO [Shantz e Schmidt, 2001] e o Adapt [Fitzpatrick et al 1998] adicionam suporte a QoS em CORBA, entretanto não possuem características de tempo-real. Já o CAVERNSoft G2 [Park 2000] é um ambiente de desenvolvimento de aplicações de realidade virtual tele-imersivas e colaborativas em rede, mas não utiliza

---

[1] PHB significa *Per-Hop Behavior* e define o tratamento dado aos pacotes em roteadores DiffServ.

[2] O termo “testbed” foi utilizado no lugar de “ambiente de teste” por já ser um termo bastante conhecido.

QoS em rede. Dessa ferramenta foi possível extrair as necessidades desse tipo de aplicação para a arquitetura do QoSWare.

No que se refere às aplicações ditas avançadas, tais como jogos de ação multiusuários [Diot e Gautier, 1999], [Lowe 2000], [Pantel 2002], [Schaefer 2002], e realidade virtual compartilhada [Leigh et al, 2001], [Johnson, 1997], esta pesquisa procurou identificar as necessidades concernentes a estas aplicações para possibilitar que as mesmas operem satisfatoriamente na Internet. Jogos de ação em rede têm a característica de enviar pacotes pequenos na rede (em geral, mensagens de posição e/ou controle), mas serem bastante sensíveis a atrasos e perdas de pacotes. A consistência da informação baseia-se no sincronismo dos dados e isso deve ser garantido para o bom funcionamento da aplicação. Aplicações de realidade virtual enviam mensagens maiores, mas também são sensíveis a atrasos e perdas de pacotes na rede.

Existem diversas técnicas para os trabalhos referentes à predição de movimento. Em [Pantel 2002], é feito um estudo de vários métodos para predizer a próxima posição do usuário para vários tipos de jogos de ação. Concluiu-se que jogos de corrida, jogos de esportes e jogos de ação em primeira pessoa (*First Person Shooting Games*) possuem resultados melhores ou piores dependendo da técnica escolhida. Em [Capin et al 1998] um algoritmo para predição de movimento para figuras humanas em aplicações de realidade virtual é apresentado e em [Meyer 2002] é proposto um modelo de séries temporais para estimar a próxima posição. Em [Smed et al 2001, 2002] são discutidas não apenas a questão da predição de movimento, como também outros aspectos referentes às necessidades de jogos multiusuários em rede.

3 Arquitetura QoSWare

A arquitetura do *middleware* QoSWare é apresentada na Figura 1 e as suas interfaces são apresentadas na Figura 2. De acordo com esta arquitetura, a aplicação se comunica com o *middleware* através de uma API (*Application Program Interface*), dividida em duas partes: uma biblioteca que a aplicação usa para poder acessar o *middleware* e um programa que torna as interfaces dinâmicas para cada aplicação.

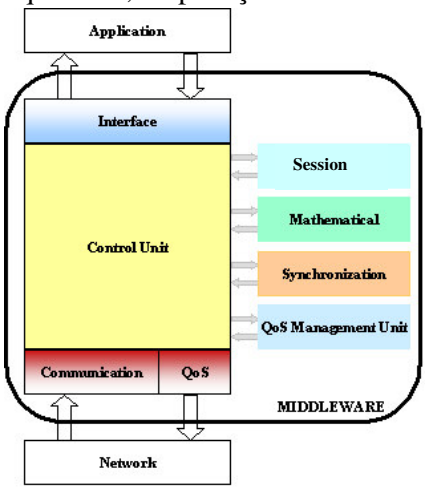


Figura 1. Arquitetura QoSWare

Existem no QoSWare interfaces dinâmicas e estáticas: as permanentes (por exemplo, as de registro da aplicação) são implementadas como bibliotecas; as dinâmicas são criadas pelo *middleware* quando a aplicação requer seus serviços. Essas interfaces invocam o módulo da Unidade de Controle quando necessário (envio/recepção de dados).

A Unidade de Controle (*Control Unit*) é responsável pela manutenção da consistência entre todos os computadores que executam a aplicação

e por garantir que todos os outros módulos estejam “trabalhando” de acordo com seus parâmetros/configurações já definidos. O controle da comunicação entre cliente e servidor é feito através da interface *mid_connection* (como apresentada na Figura 2).

O módulo de Comunicação (*Communication Unit*) possui interfaces e protocolos de rede, sendo responsável pela fragmentação da mensagem e envio dos pacotes na rede. O tipo de protocolo e a porta a serem utilizados na comunicação podem ser especificados pela aplicação quando requisita o serviço ou ficar a cargo do QoSWare decidir, baseado no tipo de serviço solicitado pela aplicação. Toda a comunicação é feita usando *sockets*. Nessa versão do projeto, apenas comunicação *unicast* foi adotada. Em breve, uma nova versão disponibilizará comunicações *multicast*, *anycast* e *broadcast* adicionadas ao *middleware.*

O componente de QoS que trabalha em conjunto com o módulo de Comunicação deve ajustar os níveis de QoS requeridos pela aplicação. Ele está fortemente relacionado ao módulo de Gerenciamento de QoS (*QoS Management Unit*) que é responsável pela negociação, reserva e controle do QoS na rede.

O Gerenciador de QoS deve ser capaz de requisitar ao provedor de Internet (ISP - *Internet Service Provider*) do cliente a reserva necessária para o bom funcionamento da aplicação e notificar à aplicação quando essas reservas não puderem ser atendidas.

Na realidade, o Gerenciador de QoS do QoSWare não suporta ajuste dinâmico dos parâmetros de QoS. Isto é, o *middleware* apenas solicita ao provedor a reserva de recurso e aguarda a confirmação ou não desta reserva. A solicitação é feita através da interface *mid_ISP* (vide Figura 2).

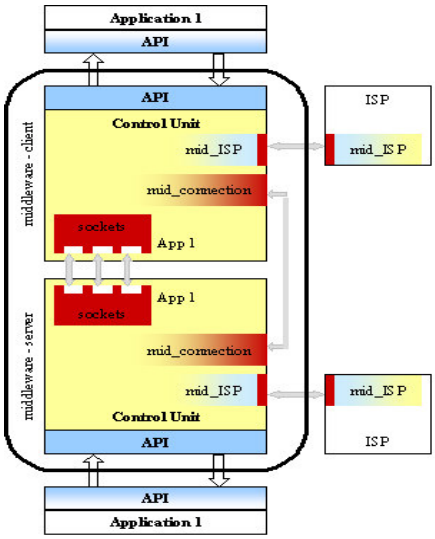


Figura 2. Interfaces do QoSWare.

Caso seja necessário (ou desejado) algum ajuste do nível de QoS, isso deve ser requisitado pela aplicação por solicitação explícita do usuário, uma vez que essa alteração também possui impacto financeiro do serviço utilizado. O *middleware* deverá prover informações sobre o estado da rede para a aplicação, permitindo que o usuário monitore se as métricas negociadas estão sendo atendidas e possa, baseado nesses valores, solicitar alterações da QoS ao seu provedor. Para tanto, o QoSWare disponibilizará à aplicação dados referentes às métricas de QoS tais como atraso, perda de pacotes, etc. De posse dessa informação, o usuário poderá ou não renegociar sua qualidade de serviço.

O QoSWare oferece três módulos com o objetivo de fornecer serviços de apoio à aplicação, são eles: sessão, matemático e sincronização. O módulo de sessão (*Session module*) é responsável pelas tarefas relacionadas ao controle da aplicação (tanto do lado do cliente quanto do servidor), tais como controle de sessão, controle dos usuários da aplicação, controle da comunicação entre servidores (por exemplo, consultas a

servidores-espelhos), controle da comunicação com o servidor máster (para gerar consultas automáticas, reportar erros, etc.) e controle de persistência em aplicações como realidade virtual compartilhada.

O módulo matemático (*Mathematical module*) implementa funcionalidades tais como predição de movimento e monitoramento do atraso médio. Por exemplo, algoritmos de interpolação e extrapolação são usados e incrementados com informações adicionais tais como ângulos e funções que definem movimentos complexos e podem ser utilizados por aplicações como realidade virtual para suavizar o movimento de figuras humanas, por exemplo. A predição de movimento permite que mesmo com atrasos e perdas de pacotes na rede, o movimento seja contínuo e suave – o que torna a aplicação mais agradável aos olhos do usuário. Mesmo que o atraso na rede possa ser controlado, ainda assim ele pode ocorrer porque há distâncias diferentes entre clientes e servidor, não dependendo da velocidade de conexão nem do equipamento de transmissão; em outras palavras, atraso é inevitável e deve ser gerenciado pelo *middleware*. O modulo de Sincronização (*Synchronization module*) mapeia o sincronismo do usuário com o sincronismo do mundo real (rede).

O QoSWare foi desenvolvido utilizando a linguagem C++ e atualmente os módulos de Interface, Unidade de Controle, Unidade Gerenciadora de QoS e Comunicação estão com todas as suas funcionalidades implementadas. Os módulos de sincronismo e sessão estão parcialmente implementados. O modulo matemático já possui implementados alguns algoritmos para predição de movimento e para convergência de posição.

3.1.Descrição do algoritmo de predição

Um algoritmo de predição simples foi utilizado para os testes descritos neste trabalho. Esse algoritmo servirá para avaliar as condições de jogabilidade descritas no item 4.3. Cabe a essa aplicação a decisão de fazer (ou não) uso desse algoritmo em ambientes com e sem qualidade de serviço.

Quando um dado não é recebido da rede num tempo hábil para atualização do jogo, o método de predição de movimento disponível no *middleware* pode ser utilizado pela aplicação para dar a impressão ao usuário do movimento do adversário. O método *predict( )* calcula qual seria a próxima posição baseado num histórico das posições anteriores. Quando a posição correta "chega", o método *isValidPrediction( )* pode ser chamado para checar se a posição prevista é válida. Se verdadeiro, então o objeto pode simplesmente "pular" para a próxima posição, corrigindo seu estado atual. Caso contrário, o método *converge( )* pode ser chamado para calcular posições intermediárias entre o valor predito e o real, criando uma suavização do movimento. O método *movement.isValidPrediction( )* depende de um limiar pré-definido para ser capaz de checar se o movimento predito é válido. Esse limite deve ser informado pela aplicação. Ele indica a quantidade máxima de "saltos", ou melhor, quantas atualizações de tela devem ser feitas até que o movimento do jogador adversário seja novamente o real.

Velocidade e aceleração são os parâmetros usados pelo método *predict( )* para calcular a nova posição provável. Pode ser um modelo polinomial de primeira ou de segunda ordem. Nos testes aqui apresentados, foi adotado um modelo polinomial de primeira ordem, considerando a velocidade dos jogadores constante (aceleração nula).

$$S_{x,y} = So_{x,y} + v_{x,y} * t \quad (1)$$

onde $S_{x,y}$ é a posição calculada nas direções horizontal e vertical da tela, $So_{x,y}$ é a posição anterior, $v_{x,y}$ é a velocidade de movimentação do jogador e t é o tempo entre as duas atualizações de tela.

4 Estudo de Caso

4.1. Descrição da aplicação-teste

Para podermos avaliar o QoSWare, um jogo de tempo-real chamado Multisoccer foi desenvolvido. A idéia era fazer um jogo com 2 jogadores e vários espectadores. O jogo é uma competição envolvendo dois jogadores de futebol representados por círculos de diferentes cores. O principal objetivo desse jogo é fazer a maior quantidade de gols possíveis contra o adversário. Uma regra importante é que nenhum jogador pode invadir o campo adversário, assim ele deve jogar a bola de longe, tentando fazer gols no outro campo. A bola é representada por um círculo branco pequeno. A Figura 3 ilustra uma tela do jogo desenvolvido.

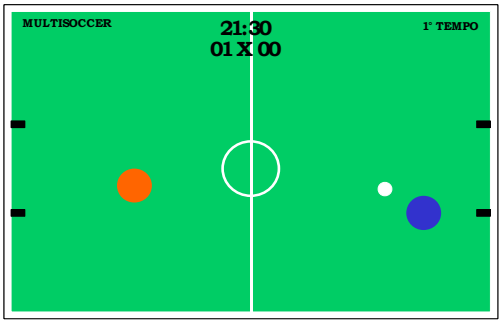


Figura 3. Tela do Multisoccer

A partida é dividida em dois períodos de tempo, cada um deles com 30 unidades de tempo. Uma unidade de tempo não necessariamente equivale a um minuto. Trata-se de um parâmetro do jogo.

A bola, ao chocar-se com um dos lados do campo, reflete sem perder a sua velocidade, tendo um movimento pseudo-aleatório (semelhante ao jogo tipo Pong [Begole e Shaffer 1997]). As velocidades do jogador e do lançamento da bola aumentam e/ou diminuem de acordo com a distância entre o jogador e o alvo (ponto clicado na tela - com o mouse - pelo usuário para servir de destino do jogador). Quanto maior for a distância, maior a velocidade.

4.2. Descrição dos cenários de teste

Os cenários de teste implementados estão descritos na Tabela 1. Os cenários A, B e C são para validar a eficiência do *middleware*. Em especial, serviram para avaliar se o middleware aumentou o consumo de CPU e memória das máquinas dos clientes e do servidor, avaliar o uso de QoS e o seu impacto na aplicação. No cenário D, as aplicações usaram o recurso de predição de movimento do *middleware*, mas em ambientes sem QoS. Para esses testes, variaram-se os parâmetros de atraso e perda no *testbed*.

Os cenários E, F e G avaliam a predição de movimento apenas no servidor, apenas no cliente e em ambos, respectivamente, em redes com diferentes níveis de QoS (PHB AF das redes DiffServ). O intuito desses últimos três cenários foi identificar condições de QoS menos exigentes quanto à perda de pacotes, mas que mantenham a jogabilidade do Multisoccer. Para tanto, variaram-se as perdas de 3% a 36%, mantendo-

se um atraso de 50ms por link. Por restrições da arquitetura do jogo, níveis de atraso não foram avaliados.

Tabela 1. Cenários selecionados para os testes

Cenário	Descrição
Cenário A	Aplicações executando sem o *middleware*;
Cenário B	Aplicações executando com o *middleware,* mas sem fazer uso de QoS;
Cenário C	Aplicações executando com o *middleware* e com suporte a QoS;
Cenário D	Aplicações executando com o *middleware* e com suporte à predição de movimento em rede sem QoS;
Cenário E	Aplicações executando com o *middleware* e com suporte à predição de movimento no servidor em rede com QoS;
Cenário F	Aplicações executando com o *middleware* e com suporte à predição de movimento nos clientes em rede com QoS;
Cenário G	Aplicações executando com o *middleware* e com suporte à predição de movimento no servidor e nos clientes em rede com QoS;

4.3.Critérios de Avaliação dos Cenários

O desempenho dos cenários acima descritos foi medido com base na jogabilidade dos usuários em cada um deles. A avaliação da jogabilidade foi feita através de um questionário onde os participantes deveriam, ao fim de cada experimento, identificar sua satisfação quanto à qualidade da partida. Foram disponibilizadas 5 opções: muito ruim, ruim, médio, bom e muito bom. Essas opções foram pontuadas de 1 a 5, respectivamente. Ao fim de cada cenário (entenda-se um conjunto de 5 partidas), foi extraída a média do cenário para cada jogador e a média geral dos jogadores. Quanto mais alta for a média, melhor terá sido a percepção/avaliação da qualidade do jogo pelos avaliadores.

4.4. Configuração do *testbed*

A Figura 4 apresenta a topologia do *testbed* utilizado. Um emulador de rede foi escolhido (no lugar de um simulador ou de uma medição real na Internet) para trazer mais confiabilidade e repetibilidade nos resultados. Assim, foi necessário fazer uso de uma ferramenta de emulação de rede chamada NIST.Net [Nist.Net 2003], uma ferramenta para configurar duas máquinas como roteadores DiffServ e um gerador de tráfego (o Traffic Generator [TG 2003]) para inserir tráfego sintético na rede repetindo o comportamento da Internet. O gerador foi parametrizado para gerar tráfego auto-similar de segundo plano para os testes.

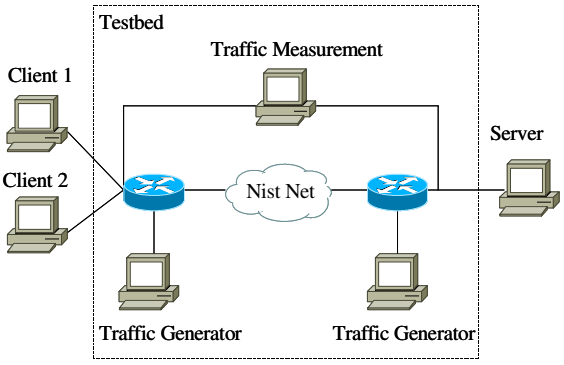


Figura 4. Topologia do *Testbed*

Para os testes apresentados nos cenários A, B, C e D desse artigo, 09 computadores tipo PC[3] foram utilizados de acordo com a topologia da Figura 4. Cada "roteador" e o emulador de rede executam em diferentes computadores. Um *switch* dedicado foi utilizado para conectar todas as máquinas.

Os geradores de tráfego criam tráfego UDP usando agregação de fontes com cauda pesada. Esses *links* possuem capacidade de 2Mbps no *uplink* e no *downlink*.

O Software TCPDump [TCPDump 2003] foi utilizado na máquina de medição de tráfego. Ele captura os pacotes de ambos os lados da rede e permite que se meça a taxa de perda de pacotes, o *delay* e o *jitter*. As aplicações executam na plataforma Windows, mas todos os outros equipamentos executam na plataforma Linux.

5 Resultados obtidos

Para avaliação dos resultados, foi desenvolvida uma ferramenta chamada UDPStat [Barbosa 2003] para extrair as informações/ estatísticas quanto às métricas de qualidade de serviço (atrasos máximo, médio e mínimo, perda de pacotes e *jitter* máximo e mínimo). O UDPStat tem como entrada dois arquivos gerados pelo TCPDump e foi implementada em C++. Os resultados são plotados com um nível de confiança de 90%.

A Tabela 2 resume os resultados obtidos nos primeiros quatro cenários apresentados no item 4.2. Foi observado que o atraso médio ficou muito próximo aos parâmetros do NIST.Net (para esses testes foi considerado um atraso de 40 ms, por ter sido um parâmetro de concepção do jogo). Notou-se também que a taxa de perda de pacotes e *jitter* tornaram-se insignificantes quando os pacotes do jogo foram marcados com alta prioridade (uso do PHB EF no cenário C). Os valores apresentados foram medidos do lado dos clientes.

Como esperado, os comportamentos dos cenários A e B foram equivalentes. Confirmando, assim, que o *middleware* fez a aplicação funcionar com sobrecargas semelhantes no processador nos cenários onde a comunicação foi implementada e controlada diretamente pelo Multisoccer. O cenário D confirmou que a predição de movimento auxilia a jogabilidade (a ser discutido mais adiante), mas não interfere na comunicação entre as máquinas.

Tabela 2. Comparação entre cenários.

	Atraso médio (ms)	Perda média de pacotes (%)	*Jitter* máximo (ms)
Cenário A	40,54	36,07%	10,88
Cenário B	40,74	49,35%	9,36
Cenário C	40,26	0,75%	0,88
Cenário D	40,59	43,63%	11,76

A Figura 5 apresenta uma amostra do tráfego recebido pelo servidor em cenários com e sem QoS. É importante destacar que o NIST.Net foi configurado para ter largura de banda limitada em 2Mbps em ambas as direções. Observa-se que, como esperado, no cenário com QoS, o tráfego foi mais "comportado".

[3] As máquinas do servidor e dos clientes possuem processadores Pentium 4 1.5GHz e 512MB RAM. As demais máquinas do *testbed* possuem processadores Athlon 1.5GHz e 256MB RAM.

Para medir a carga de processamento das máquinas que executavam os clientes e o servidor, o software Performance (disponível no Windows 2000 Server) foi utilizado. Foi observado que o *middleware* não comprometeu o desempenho da aplicação no que se refere ao consumo do processador. A Tabela 3 apresenta os valores médios obtidos nos testes dos cenários A, B e C, tanto no servidor como em um dos clientes. Como o QoSWare praticamente só faz uso de memórias estáticas, o consumo de memória não variou significativamente.

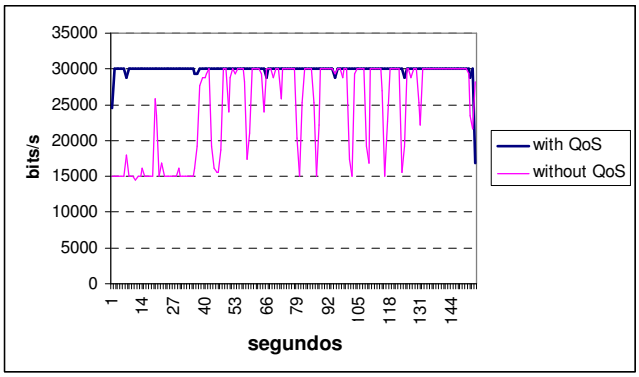


Figura 5. Tráfego que chega ao servidor em cenários com e sem QoS.

Tabela 3. Tempo de ocupação do processador pela aplicação (%)

Tempo de ocupação do processador (%)	Servidor	Cliente 1
Cenário A	98,68	96,73
Cenário B	97,94	97,88
Cenário C	97,68	99,36

Os gráficos apresentados na Figura 6 ilustram os resultados das partidas (cenários C, E, F e G citados no item 4.2) utilizando os critérios de avaliação descritos no item 4.3. A Figura 6(a) ilustra os resultados para um cenário C (sem predição de movimento), mas utilizando o PHB AF em vez do PHB EF. Vale relembrar que os jogadores possuíam prioridades em relação aos outros tráfegos, mas variou-se o percentual de perda de pacotes admissível para esses dados. Admitindo-se que a jogabilidade estará comprometida a partir do momento onde o jogo passa a ser avaliado como ruim, os limites de percentual de perda de pacotes encontrados em cada um desses cenários foram de: 6% para os cenários sem predição de movimento; 9% para os cenários com predição apenas no servidor; 12% para os cenários com predição apenas nos clientes; e 27% com predição de movimento tanto no servidor quanto nos clientes.

Quando um pacote de posição é perdido em um jogo sem predição, o jogador aparece parado na tela do adversário e quando um novo pacote de posição é recebido, a posição é então atualizada implicando em "saltos" na tela. Houve mais "saltos" nos cenários sem predição do que nos cenários utilizando o algoritmo de predição de movimento (descrito na seção 3.1). Nos cenários onde se acrescentou predição de movimento nos clientes e no servidor (gráficos apresentados nas Figuras 6(b) e 6(c)), os jogadores conseguiram continuar jogando em cenários com percentual de perdas maiores que as apresentadas na Figura 6(a). Comparando-se as Figuras 6(b) e 6(c) observa-se que a avaliação dos cenários com predição de movimento nos clientes foi um

pouco melhor do que as dos cenários com a predição inserida apenas no servidor. Isso pode ser explicado pelo fato de que com a predição no servidor garante-se uma "correção" de posição caso o pacote enviado pelo cliente não chegue ao servidor, todavia se houver perdas de pacotes no sentido contrário continuará havendo "saltos" na tela dos jogadores.

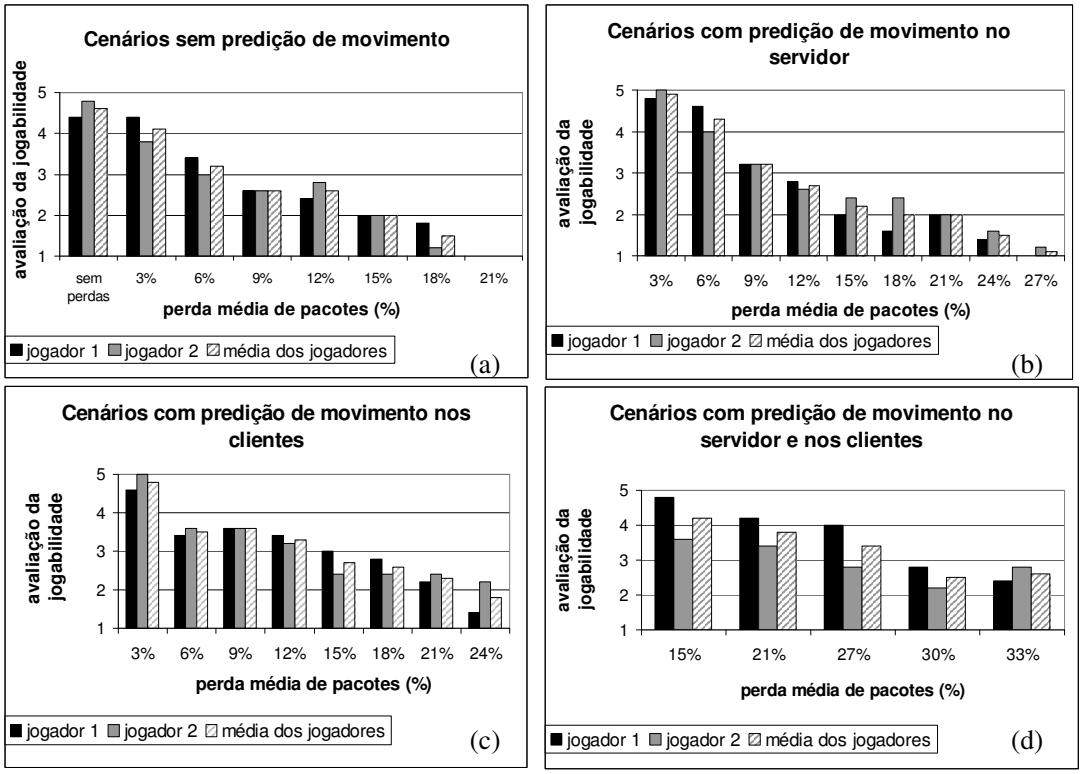


Figura 6. Avaliação da jogabilidade em redes com diferentes níveis de QoS.

Quando a predição passou a ser feita nos clientes, o movimento dos jogadores foi suavizado, tornando o jogo mais agradável. Todavia, em alguns momentos "saltos" voltaram a acontecer. Esse fato ocorreu em situações do tipo: o servidor não recebeu o pacote enviado por um ou mais clientes e retornou a estes as ultimas posições que ele tinha armazenado de um ou mais jogadores. Digamos que o pacote de retorno, enviado pelo servidor, também tenha sido perdido. Assim, o cliente faz a predição de movimento e desloca o adversário na sua tela. Numa nova atualização de tela, o cliente recebe uma nova posição e, caso a sua predição não tenha sido eficiente, um "salto" deverá ocorrer para a correção dessa nova posição do outro jogador. No cenário apresentado na Figura 6(d), a movimentação dos jogadores ainda ficou contínua e pouquíssimos saltos ocorreram ao longo da partida. Observa-se também que nesse último cenário foi possível continuar jogando em redes onde a perda de pacotes foi considerada alta.

A Figura 7 apresenta dois gráficos referentes ao erro do modelo de predição adotado em uma configuração de perda de pacotes a 15%. A Figura 7(a) ilustra o erro máximo em pixels e a Figura 7(b) o erro médio encontrado em cada cenário. Em cada cenário foram mais de 19.000 pacotes (isto é, posições) recebidos pelos clientes. O erro médio ficou muito baixo porque os jogadores permaneceram por muito tempo parados,

esperando que o adversário lançasse a bola. Conforme apresentado na Figura 7(b), o erro máximo diminuiu significativamente nos cenários com predição nos clientes e no servidor, o que justifica a diminuição dos "saltos" na tela.

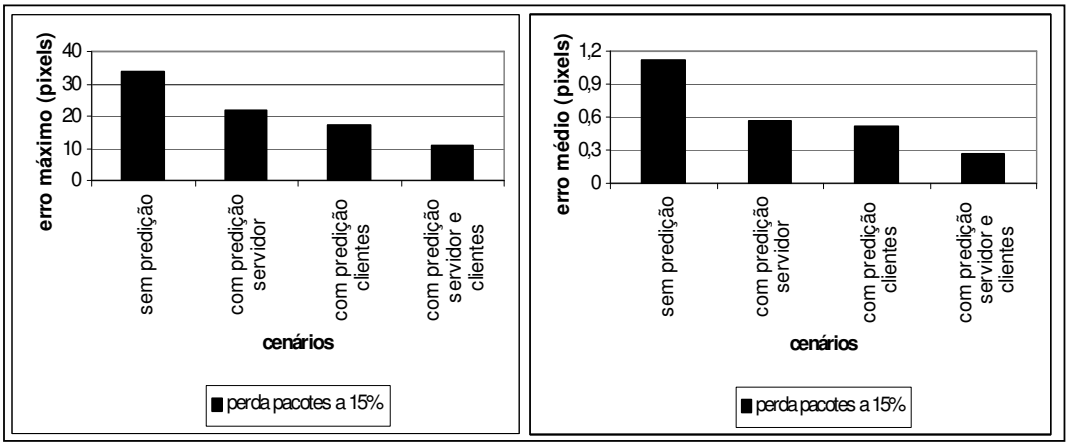


Figura 7. Comparação entre os erros máximos e médios dos diferentes cenários.

A Figura 8 apresenta uma comparação dos erros médios e máximos dos cenários sem predição, com predição apenas no servidor e com predição apenas nos clientes. Por sua vez, na Figura 9 são apresentados os erros médios e máximos do cenário com predição tanto nos clientes como no servidor. Essa última figura foi apresentada separadamente em função dos valores de perda de pacotes alcançados nesse cenário serem maiores que nos demais.

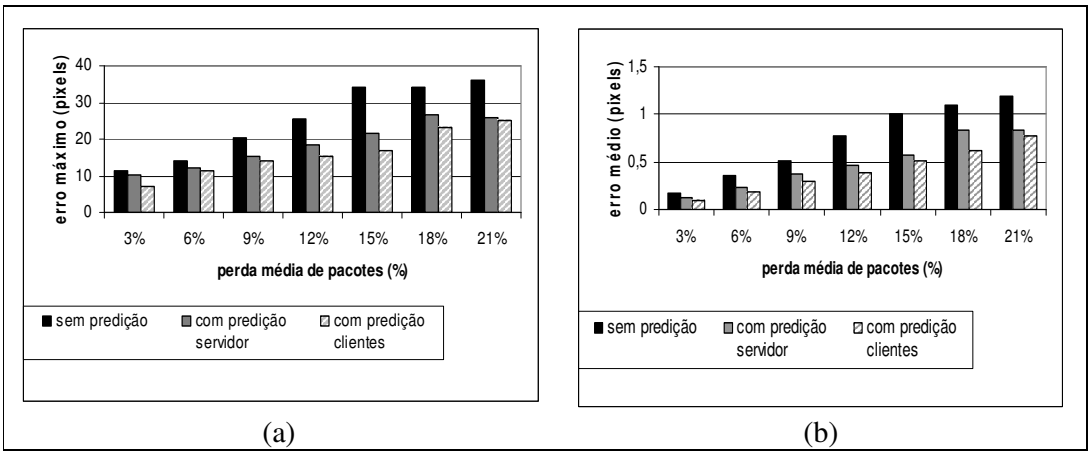
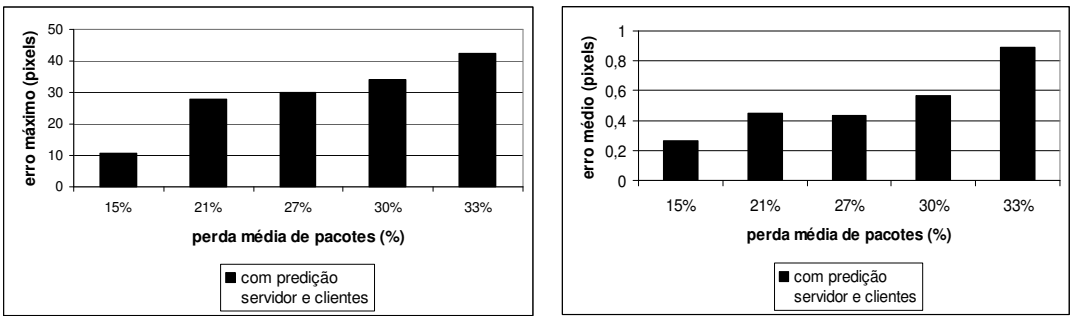


Figura 8. Comparação entre erros médios e máximos em vários níveis de QoS para os cenários sem predição de movimento, com predição de movimento apenas no servidor e com predição de movimento apenas nos clientes.

Comparando os gráficos das Figuras 6 e 8(a), observa-se que os jogadores passaram a avaliar a jogabilidade da aplicação como sendo de nível médio (valor igual a 3 no gráfico) quando o erro máximo ficou em torno de 15 pixels em todas as três situações referentes à predição. Observando a Figura 9 percebe-se que o erro máximo aumentou em relação aos dados da Figura 8, todavia o percentual de perda de pacotes

também aumentou significativamente. Assim, o uso da predição contribuiu para que os jogadores não sentissem os efeitos da perda de pacotes e quando a predição de movimento foi inserida tanto no servidor quanto nos clientes permitiu uma diminuição na quantidade de "saltos" ou ajustes de posição, permitindo que os jogadores continuassem satisfeitos, mesmo com níveis de QoS menos exigentes.



(a) (b)

Figura 9. Erros médios e máximos para diferentes níveis de QoS nos cenários com predição de movimento no servidor e nos clientes.

6 Considerações finais e trabalhos futuros

Procura-se com o *middleware* QoSWare construir um ambiente independente de plataformas de suporte a sistemas distribuídos tais como Java/RMI, CORBA (ou suas versões como o Real-Time CORBA) ou DCOM. Todavia acredita-se que ele seja capaz de adaptar-se/integrar-se a essas plataformas caso seja necessário. O QoSWare se propõe a ser um *middleware* que não gera sobrecarga de processamento nem consumo excessivo de memória nas máquinas, bem como não compromete a necessidade de tempo-real da aplicação. O QoSWare não pretende ser uma plataforma de desenvolvimento como as supra citadas, mas sim prover comunicação, negociação de QoS e marcação dos pacotes, funcionando como um *toolkit* que pode ser integrado às aplicações.

O QoSWare consegue promover transparência de comunicação para a aplicação de forma simples e estruturada, e mais que isso, permite que pacotes atravessem a rede com suporte a QoS (DiffServ), dando prioridade aos mesmos, segundo o que tiver sido acordado entre o usuário da aplicação e seu provedor de rede.

Durante o estudo de caso, foi observado que aplicações de tempo-real promovem pacotes pequenos para serem enviados na rede, atraso e perda de pacotes são de fato um problema para esse tipo de aplicação. Apesar disso, observou-se que predição de movimento quando adicionada ao *middleware* melhorou consideravelmente a jogabilidade em redes sem QoS, mas praticamente não foi necessário em redes com QoS que utilizavam o PHB EF.

Alternado os níveis de QoS, observou-se que a predição de movimento seja apenas no servidor, ou apenas nos clientes ou em ambos, permitiu que os jogadores continuassem satisfeitos com a qualidade da aplicação mesmo com percentuais de perda de pacotes mais altos. Isso permitirá que os usuários paguem menos por um serviço que *a priori* necessitaria de um nível de QoS bem superior. Observou-se também que

quando se faz uso da predição de movimento tanto no servidor quanto nos clientes, o erro da predição diminui mais do que se tivesse sido implementado apenas no servidor ou apenas nos clientes.

Alguns trabalhos futuros incluem a comparação de outros algoritmos de predição, a implementação da comunicação e da negociação de QoS com o provedor de serviço de Internet (*Internet Service Provider* -ISP), testar o QoSWare com aplicação de realidade virtual e implementar prioridade/seletividade de pacotes para uma mesma aplicação, por exemplo, em jogos de aplicação com *chat*, onde os pacotes do jogo devem ter prioridade maior que os de texto.

Um outro requerimento que será implementado num próximo passo diz respeito à segurança de dados, tais como técnicas de criptografia. Dispositivos com espaço de memória e resolução de vídeo diferentes que participam da mesma aplicação devem receber, no futuro, dados com tamanhos diferentes e condizentes com as suas limitações físicas, sem que isso, entretanto, traga prejuízos à confiabilidade da aplicação e ao requisito de tempo-real: o QoSWare também será capaz de gerenciar esse tipo de situação, desde que a aplicação envie os diferentes pacotes para ele.

7 Agradecimentos

Os autores agradecem ao CNPq pelo financiamento do projeto QoSWare.

8 Referências bibliográficas

Aiken, B. et al (2000) "Network Policy and Services: A Report of a Workshop on Middleware", RFC 2768, February.

Bakken. D. (2001) "Middleware". Encyclopedia of Distributed Computing, Kluwer Academic Press.

Barbosa, R. "UDPStat" , http://www.cin.ufpe.br/~rsbgb/udpstatNew/. Última visita em novembro, 2003.

Begole, J. e Shaffer, C. (1997) "Internet Based Real-Time Multiuser Simulation: Ppong!" Technical Report TR-97-01. Virginia Poly- technic Inst. and State University, Department of Computer Science, Fevereiro.

Capin, T., Esmerado, J. e Thalmann, D (1997) "A Dead-Reckoning Technique for Streaming Virtual Human Animation", Proc. VRAIS '97, pp. 161-169.

Diot, C., e Gautier, L. (1999) "A distributed architecture for multiplayer interactive applications on the internet", *IEEE Network*. vol. 13, n° 4, pp. 6-15.

Fitzpatrick, T.; Blair, G., Coulson, G.; Davies, N. e Robin, P. (1998) "Software Architecture for Adaptive Distributed Multimedia Applications" IEEE Proceedings – Software. Vol. 145, No. 5, Outubro.

Internet2 Consortium. "Middleware". http://middleware.internet2.edu/index.html. Última visita em novembro, 2003.

Johnson, A. E.; Leigh, J.; Defanti, T. A. (1997) "Issues in the design of a flexible distributed architecture for supporting persistence and interoperability in

collaborative virtual environments", University of Illinois at Chicago, Technical Report, 14pp, Chicago.

Kon, F.; Román, M.; Liu, P.; Mao, J.; Yamane, T.; Magalhães, L. C. e Campbell, R. (2000) "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB", IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000). New York, Abril.

Leigh, J.; Yu, O.; Schonfeld, D.; Ansari, R.; He, E.; Nayak, A.; Ge, J.; Krishnaprasad, N.; Park, K.; Cho, Y.; Hu, L.; Fang, R.; Verlo, A.; Winkler, L.; Defanti, T. A. (2001) "Adaptative networking for tele-immersion". Procc. Immersive Projection Technology/Eurographics Virtual Environments Conference, Stuttgart, Germany,

Lowe, N. e Strauss, J., (2000) "Realtime 3D Multiplayer Internet Gaming", University of Western, Australia.

Meyer, D. (2002) "Naive Time Series Forecasting Methods", R News, Vol.2/2, p. 7-10, Junho.

NIST.Net. http://snad.ncsl.nist.gov/itg/nistnet. Última visita em novembro, 2003.

Pantel, L. e Wolf, C. L. (2002)"On the Impact of Delay on Real-Time Multiplayer Games", International Workshop on Network and Operating System Support for Digital Audio and Video: Network Issues for Video and Games, New York, NY, USA, ACM Press, p. 23-29.

Park, K.; Cho, Y. J.; Krishnaprasad, N.; Scharver, C.; Lewis, M.; Leigh, J.; Johnson, A. (2000) "CAVERNSoft G2: a toolkit for high performance tele-immersive collaboration". In: ACM Annual Symposium On Virtual Reality Software & Technology, 7, Seoul, Korea, Proceedings. p.8-15.

Schaefer, C. e Enderes, T. e Ritter, H. e Zitterbart, M. (2002) "Subjective Quality Assessment for Multiplayer Real-Time Games", ACM NetGames, Braunschweig, Germany, p. 74-78.

Schantz; R. e Schmidt, D. (2001) "Middleware for Distributed Systems - Evolving the Common Structure for Network-centric Applications". The Encyclopedia of Software Engineering. John Wiley & Sons. Dezembro.

Smed J., Kaukoranta T. e Hakonen H. (2001) " Aspects of Networking in Multiplayer Computer Games". Proceedings of International Conference on Application and Development of Computer Games in the 21st Century.

Smed, J., kaukoranta, T. e Hakonen, N. (2002) "A Review on Networking and Multiplayer Computer Games", TUCS Technical Report No. 454, Abril.

Stiller, B., Class, C., Waldvoge, M., Caronni, G., Bauer, D. (1999) "A flexible *middleware* for multimedia communication: design, implementation and experience", IEEE Journal on Selected Areas in Communication. Vol. 17, n° 9, Setembro.

TCPDump. http://www.tcpdump.org. Última visita em novembro, 2003.

TG - Traffic Generator. http://www.postel.org. Última visita em novembro, 2003.