Qualidade de Serviço Multinível baseada em ATM para a Plataforma Multiware

Flávio Henrique de S. Lima
fhlima@dcc.unicamp.br

Edmundo R. M. Madeira
edmundo@dcc.unicamp.br

Instituto de Computação
Universidade Estadual de Campinas
Caixa Postal 6176 CEP 13083-970
Campinas-SP

Resumo

Este artigo apresenta um modelo de Qualidade de Serviço de múltiplos níveis, desenvolvido para a plataforma Multiware. Tal modelo atua em diversos níveis de protocolos, de forma a garantir a Qualidade de Serviço das aplicações. O modelo foi desenvolvido para redes ATM, onde alguns requisitos de QoS são garantidos pela rede. Como protocolo de transporte escolhemos o *XTP (Xpress Transport Protocol)*, devido às suas características de tempo real. Neste trabalho portamos o protocolo XTP para redes ATM, e o adaptamos para suportar requisitos de QoS. Para que o modelo suportasse aplicações distribuídas, como exigido pela plataforma Multiware, desenvolvemos um Objeto de Negociação Distribuída de QoS, baseado no modelo ODP e na arquitetura CORBA. Todo o modelo foi implementado usando uma linguagem orientada a objeto, compatível com implementações da arquitetura CORBA. Desta forma, aplicações distribuídas e orientadas a objeto podem especificar e negociar requisitos de QoS em um ambiente padronizado.

Abstract

This paper presents a Quality of Service model with multiple levels, developed for the Multiware Platform. Such model acts in many protocol levels, in order to guarantee the Quality of Service of the applications. This model was developed for ATM networks, where some QoS requirements are guaranteed by the network. As transport protocol we chose XTP (*Xpress Transport Protocol*), due to its real-time characteristics. In this work we have ported XTP to ATM networks, and we adapted it for supporting QoS requirements. As demanded by the Multiware platform, the model has to support distributed applications. For this, we developed a Distributed QoS Negotiation Object, based on the ODP model and the CORBA architecture. All the model was implemented using an object-oriented language, compatible with CORBA implementations. In this way, distributed object-oriented applications can specify and negotiate QoS requirements in a standardized environment.

1 Introdução

Nos últimos anos temos visto um aumento considerável na velocidade das redes de computadores, com o surgimento de redes de alta velocidade, como FDDI, Fast Ethernet ou ATM. Tal aumento de velocidade proporcionou que uma gama de novas aplicações, antes inviáveis por restrições de desempenho, fosse criada. Entre essas aplicações estão: video-conferência, multimídia em rede, telemedicina, realidade virtual e outras. Além dessas novas aplicações, aquelas já comuns passaram a incorporar grandes volumes de dados, explorando os novos limites de velocidade.

Essas novas aplicações possuem em comum o fato de lidarem com grandes quantidades de dados, e de terem restrições temporais. Tais aplicações necessitam que a rede lhes garanta um conjunto de requisitos de comportamento, conhecidos coletivamente como Qualidade de Serviço (QoS - *Quality of Service*).

As aplicações mais conhecidas, como as de transação, por sua vez passaram a ter um caráter distribuído, e com isto passaram a necessitar de garantias quanto a segurança, disponibilidade, etc. Tais garantias devem ser providas pelo ambiente distribuído no qual a aplicação executa. Esses ambientes distribuídos, por sua vez, incorporaram novos serviços, seguindo modelos de distribuição como o RM-ODP (*Reference Model for Open Distributed Processing*) [1,2,3]. O advento da arquitetura CORBA (*Common Object Request Broker Architecture*) contribuiu para viabilizar o uso de plataformas distribuídas em larga escala, padronizadas e interoperáveis [4].

Assim sendo, o aparecimento de novas aplicações e a distribuição dessas aplicações exigem da rede e do ambiente distribuído uma série de qualidades de serviço. Neste artigo introduzimos um modelo de QoS multinível para a plataforma Multiware. Tal modelo atua nos diversos níveis abaixo das aplicações, de forma a garantir que as qualidades citadas sejam providas.

Este modelo foi implementado em uma rede baseada em ATM (*Asynchronous Transfer Mode*). Os parâmetros de QoS providos pelo ATM são manipulados por um protocolo de transporte de tempo real (XTP), que foi modificado para trabalhar com QoS. Acima do XTP há um objeto responsável pela negociação de QoS no ambiente CORBA. Desta forma, foi criado um ambiente integrado para criar aplicações distribuídas, orientadas a objeto, com requisitos de qualidade de serviço e padronizadas.

Este artigo está organizado da seguinte forma: a seção 2 descreve a plataforma Multiware, onde este projeto está inserido. A seção 3 explica os diversos fatores que compõem a QoS nos diferentes níveis. A seção 4 descreve o protocolo XTP. A seção 5 trata do modelo de QoS multinível aqui proposto, enquanto a seção 6 trata dos aspectos envolvidos na implementação do modelo. Finalmente, a seção 7 apresenta algumas considerações finais sobre o projeto.

2 A Plataforma Multiware

A Multiware é uma plataforma de suporte a computação distribuída, cujo objetivo é facilitar o processo de desenvolvimento de aplicações distribuídas, como CSCW (*Computer Supported Cooperative Work*) [5]. A plataforma Multiware foi desenvolvida usando os conceitos do modelo ODP. Sua estrutura é mostrada na Figura 1.

A camada de suporte provê os protocolos de comunicação e as facilidades dos sistemas operacionais necessárias à operação em rede. Esta camada inclui protocolos baseados em chamadas remotas, como RPC (*Remote Procedure Call*).

A camada Middleware é dividida em três subcamadas: ORB, Serviços de Objetos e Processamento Multimídia. A subcamada ORB trata das funções básicas de suporte a distribuição, e atualmente está baseada em uma implementação da CORBA 2.0, a Orbix ®. Os Serviços de Objetos são um conjunto de funções comuns a várias aplicações distribuídas, como o suporte a grupos. A subcamada de Processamento Multimídia trabalha em paralelo com as outras subcamadas. Como aplicações multimídia não toleram atrasos muito grandes, nem atrasos variáveis, nestes casos a interface Object Services/ORB é somente usada no estabelecimento da conexão. Após esse passo inicial, toda a transferência de dados ocorre através da subcamada de Processamento Multimídia.

A camada Groupware provê suporte a aplicações multimídia de trabalho cooperativo, usando as facilidades da camada Middleware. Entre essas aplicações, tem sido destacado o suporte a CSCW. As aplicações CSCW cobrem um grande conjunto de aplicações, como video-conferência, co-edição, etc.

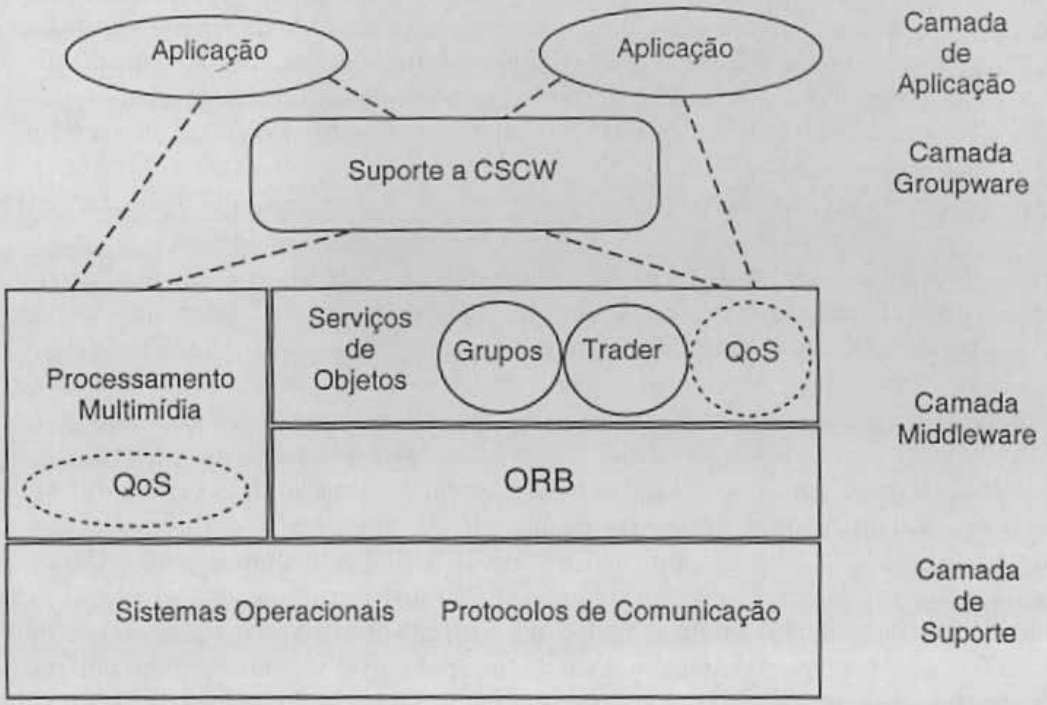

Figura 1 : Plataforma Multiware

3 Qualidade de Serviço

O modelo de referência ODP define Qualidade de Serviço como "um conjunto de requisitos de qualidade sobre o comportamento coletivo de um ou mais objetos" [3]. Em outro
contexto, Vogel define QoS como "o conjunto de características quantitativas e qualitativas de um sistema multimídia distribuído, que são necessárias para obter a funcionalidade requerida de uma aplicação" [6]. De forma genérica, Qualidade de Serviço é um conjunto de métricas de qualidade, interpretadas em diferentes níveis, que regulam o comportamento de uma aplicação [7].

Para mensurar a QoS é preciso definir o conjunto de métricas de interesse para uma determinada aplicação, e estabelecer um conjunto de valores-limite para essas métricas [8,9,10,11]. Cada métrica deve ser coletada e interpretada em seu nível apropriado. Por exemplo, as métricas de controle de fluxo devem ser interpretadas no contexto do protocolo de transporte utilizado. Há, no entanto, métricas que possuem significado sob mais de um contexto. Por exemplo, o *throughput* de uma aplicação pode ser interpretado pelo protocolo de transporte, visando a alocação de buffers, e pela arquitetura da rede, visando a reserva de recursos.

3.1 QoS no nível de rede

O nível de rede controla a entrega de dados na inter-rede. Várias características do nível de rede determinam a QoS que será entregue aos níveis superiores.

O roteamento determina o caminho que os pacotes de dados seguirão entre a fonte e o destino. A capacidade de entrega de QoS do nível de rede depende inteiramente da soma das capacidades individuais de cada elemento intermediário. Uma analogia possível seria com uma seqüência de tubos conectados, onde a vazão máxima é determinada pelo tubo mais fino. Da mesma forma, a capacidade de QoS fim-a-fim vai depender do elemento intermediário mais lento, ou que introduz o maior atraso. Assim sendo, é necessário que o algoritmo de roteamento conheça as métricas de QoS. O algoritmo de roteamento deve ser integrado com a reserva de recursos, de forma a evitar os elementos intermediários que não possam prover a QoS desejada. Um determinado nível de QoS só será provido se houver um caminho entre fonte e destino, por onde todos os elementos intermediários garantam aquele nível de QoS. Os novos protocolos de reserva de recursos, como o RSVP (*Resource Reservation Protocol*), são baseados nesse paradigma.

A reserva de buffers de transmissão e de recepção é também um elemento muito usado para garantir níveis de QoS. Mesmo que a rede não possibilite a reserva de outros recursos, é possível prover algum nível de QoS através da reserva de capacidade de transmissão e recepção. No nível de rede, esses buffers existem em cada elemento intermediário, para evitar congestionamentos e conseqüente perda de dados. Através da reserva de buffers, é possível alocar a uma determinada conexão uma certa capacidade de transmissão. Uma outra utilidade dos buffers é a compensação da variação de retardo (jitter) na entrega de pacotes. Acoplando-se um temporizador aos buffers, é possível compensar o jitter de uma conexão, que é uma importante métrica de QoS. Os buffers também têm grande importância no controle de fluxo, que é uma característica do nível de transporte.

3.2 QoS no nível de transporte

O nível de transporte controla os recursos de comunicação fim-a-fim, e também faz a interface com os níveis superiores. Muito freqüentemente, os dados de transporte são entregues diretamente às aplicações, principalmente no caso de aplicações multimídia. Veremos a seguir algumas dessas características.

O controle de erros é uma métrica importante de QoS. Muitas aplicações, como as de transação, são baseadas no paradigma de que o meio de transmissão é capaz de entregar mensagens sem erro. Sendo assim, é importante poder limitar a quantidade de erros a um valor probabilístico máximo, que vai depender das características do meio e dos algoritmos utilizados para detecção e correção de erros. Outras aplicações, como a transmissão de vídeo, podem suportar uma certa quantidade de erros, que é compensada pela redundância da informação. Uma transmissão de vídeo tipicamente transfere 24 a 30 quadros por segundo. Um erro que modifique parte de um único quadro pode ser facilmente tolerado. Além disso, cada quadro só possui valor se puder ser apresentado dentro de um determinado limite de tempo, e a correção de erros pode trazer atrasos intoleráveis.

Da mesma forma que no nível de rede, o nível de transporte exerce controle sobre buffers, sendo que neste nível os buffers são fim-a-fim. Isto possibilita um melhor controle sobre o fluxo de dados entre emissor e receptor. Muitos protocolos de transporte, conforme veremos posteriormente, possuem interfaces próprias para a alocação de buffers de transmissão e recepção.

O controle de fluxo permite evitar que a diferença de velocidade entre transmissor e receptor provoque a perda de dados. Isto é feito através de diversos mecanismos que permitem sinalizar ao emissor a proximidade de

estouro dos buffers do receptor, o que faz com que ele diminua a taxa de emissão de pacotes. No TCP, por exemplo, isto é feito através de um mecanismo conhecido como *slow start*, baseado na perda de pacotes.

O controle de taxa permite explicitamente limitar a taxa de envio de dados de um emissor. Esse controle é geralmente feito através de um mecanismo de fichas (*tokens*). O controle de taxa funciona também como um mecanismo de policiamento, ao evitar que os usuários utilizem mais recursos do que o nível contratado.

3.3 QoS no modelo ODP

Em relação à Qualidade de Serviço, o modelo ODP adota uma abordagem bastante genérica [8]. O modelo somente indica em quais pontos deve haver uma interface de QoS, e quais as relações entre a QoS e os pontos-de-vista (*viewpoints*) de computação e engenharia. O modelo também mostra que a provisão de QoS é uma propriedade essencial de sistemas contruídos de acordo com o modelo ODP [3]. Apresentaremos a seguir alguns elementos do modelo ODP e seu relacionamento com a Qualidade de Serviço.

Um contrato ODP é um acordo que regula a cooperação entre objetos [3]. Um contrato pode ter aspectos dinâmicos e estáticos. Isto é, os termos do contrato podem ser modificados durante a interação. Um contrato especifica os papéis associados aos objetos, seus relacionamentos de cooperação e o tipo de comportamento que o invalida. Além disso, um contrato pode especificar os aspectos de QoS da interação. Há uma categoria especial de contratos, chamados contratos de ambiente (*environment contracts*), onde um objeto especifica suas restrições em relação ao ambiente, e vice-versa. Essas restrições são freqüentemente atributos de QoS. O modelo ODP não define explicitamente uma notação para especificar QoS no nível de linguagem computacional.

O ponto-de-vista computacional especifica um modelo de ligação (*binding*), que regula a forma pela qual a associação entre interfaces computacionais é feita. Um *binding* cria um caminho de comunicação entre interfaces computacionais [3]. O *binding* pode ser implícito ou explícito. Quando o binding é implícito, a simples invocação de um objeto faz com que seja criada uma ligação do tipo apropriado para as interfaces. O *binding* explícito é subdividido em primitivo e composto. Um *binding* primitivo permite a ligação entre somente duas interfaces computacionais, e é iniciado por uma das entidades comunicantes. O *binding* composto permite a associação entre duas ou mais interfaces, e é controlado por um objeto especial, chamado Objeto de Binding (Figura 2).

Além de criar e manter a associação entre objetos computacionais, outra função do Objeto de Binding é tratar dos aspectos de QoS da comunicação. O Objeto de Binding tem uma interface de controle, que possibilita a

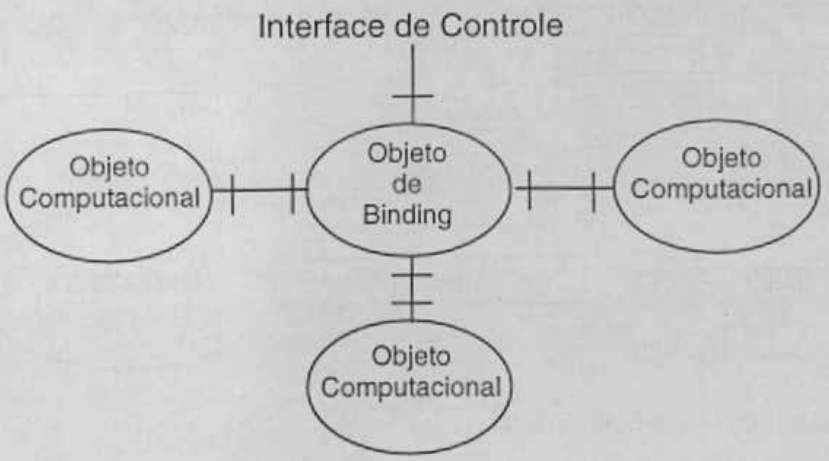

Figura 2: Objeto de Binding

negociação de QoS. Essa interface pode ser usada para notificar mudanças nos padrões de QoS, habilitando a renegociação.

A partir do ponto-de-vista de engenharia, o Objeto de Binding transforma-se nos objetos de infra-estrutura que controlam o canal, como mostrado na Figura 3. Nesta estrutura, além do Objeto de Binder, outro elemento que atua no controle da QoS é o Objeto de Protocolo. O Objeto de Protocolo usa a infra-estrutura da rede para prover as facilidades de comunicação, de acordo com a necessidade dos objetos comunicantes [3]. O Objeto de Protocolo pode ter uma interface de controle, onde os Objetos Básicos de Engenharia (BEOs - *Basic Engineering Objects*) podem especificar suas necessidades de QoS e receber notificações vindas da rede, que podem disparar ações de renegociação.

O modelo ODP define um objeto especial, chamado Trader, cuja função é mediar o anúncio e a descoberta de interfaces [12]. O Trader trabalha através de ofertas de serviço, que são um conjunto de informações sobre uma

interface. Essas ofertas de serviço são anunciadas e descobertas através dos papéis (*roles*) que um objeto pode ter: exportador e/ou importador (Figura 4).

O objeto exportador notifica o Trader de suas ofertas de serviço. O objeto importador consulta o Trader quando necessita de algum serviço. Nesse contexto, é definida também uma política de *trading*, que regula o comportamento do Trader. As informações contidas nas ofertas de serviço freqüentemente são atributos de QoS.

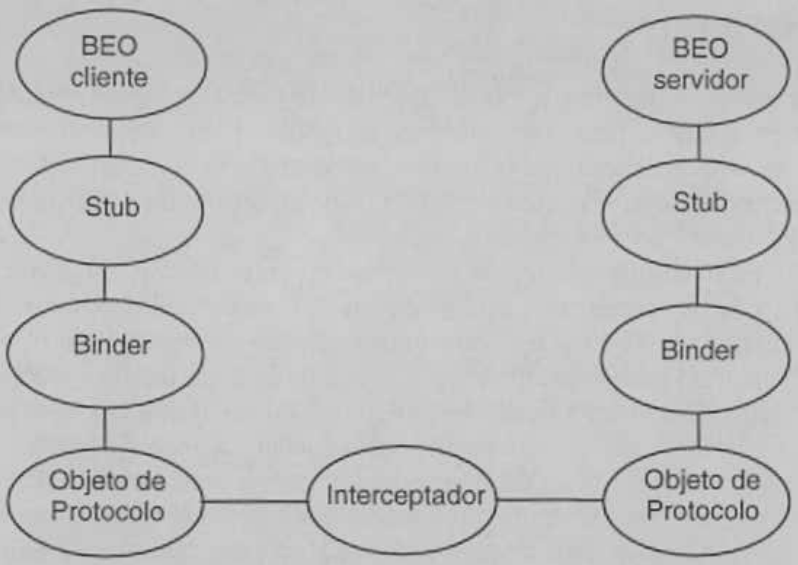

Figura 3: Objetos do Canal

Porém, nesse nível, os atributos de QoS dizem respeito a métricas de alto nível, como segurança, tolerância a falhas, etc [7].

Figura 4: Operações do Trader

3.4 Protocolos de Tempo Real

Protocolos de Tempo Real (PTRs) têm seu maior uso em aplicações multimídia, além de aplicações de tratamento de mensagens. Nas aplicações multimídia, o principal objetivo de um PTR é recuperar o sincronismo da informação transmitida ao chegar no destino. No tratamento de mensagens, o principal objetivo é priorizar a chegada de mensagens.

Para a recuperação do sincronismo, o PTR inclui informações de temporização durante a transmissão dos dados (*timestamps*). Esses *timestamps* podem conter informação de temporização relativas aos outros pacotes, ou ainda informação de tempo absoluto (*clockwall time*), ou ainda ambos. A vantagem de haver informação de tempo absoluto aparece quando diferentes mídias devem ser sincronizadas, como áudio e vídeo. Esse controle, chamado *"lip sync"*, é facilitado quando há informação de tempo absoluto.

Devido aos requisitos de sincronização, o PTR inclui uma forma de desabilitar a retransmissão de pacotes recebidos com erro. Alguns PTRs notificam a recepção de pacotes com erro, e outros nem isto fazem. Outro mecanismo comum aos PTRs é o controle de buffers. O controle de buffers é o mecanismo básico para minimizar

os efeitos do atraso e do jitter associados à comunicação. O tamanho e o número de buffers são fatores importantes para determinar a eficiência de um PTR.

Alguns PTRs proporcionam a priorização do tráfego. Por exemplo, tornam possível priorizar determinadas mídias em detrimento de outras, no caso de tráfego multimídia. No caso de uma video-conferência, se houver um congestionamento, o áudio é geralmente mais importante que o vídeo.

Outra característica desejável em PTRs é a existência de monitoração de QoS. O PTR pode avisar à aplicação quando ocorre uma degradação na Qualidade de Serviço. Nesse caso, a aplicação pode renegociar sua QoS, ou ainda adaptar-se ao novo padrão de tráfego, no caso das aplicações adaptativas.

Ainda outra característica comum aos PTRs é o controle de multicast. Os PTRs que atuam na camada de rede devem necessariamente prover esse controle, enquanto que os PTRs da camada de transporte podem utilizar o suporte a multicast provido pela camada de rede, acrescentando ainda alguma funcionalidade extra. Alguns exemplos de protocolos de tempo real são: XTP (camada de transporte), RTP (camada de transporte) e ST-II (camada de rede). O XTP será explicado na próxima seção.

O RTP (*Real Time Protocol*) é um protocolo estritamente de transporte [13]. Ele pode utilizar IP, UDP ou AAL5 como protocolos de rede. Ao contrário do XTP, o RTP é orientado a datagramas. O RTP sozinho não provê nenhum mecanismo que garanta a entrega de dados em um tempo especificado, nem provê garantias de Qualidade de Serviço, mas ele espera que os protocolos de mais baixo nível garantam isto. Ele também não garante a entrega dos dados em seqüência, e nem mesmo garante que um certo datagrama seja entregue (entrega não-confiável). O RTP foi construído para prover a informação de temporização necessária a uma aplicação, podendo ser integrado à aplicação ao invés de funcionar como uma camada separada. O RTP é um *"framework"* de protocolo propositalmente incompleto. Ele é composto basicamente de dois protocolos: além do próprio RTP, responsável pelo transporte de dados em tempo real, existe o RTPCP (RTP *Control Protocol*), que monitora a Qualidade de Serviço e entrega informações sobre a QoS para os participantes da comunicação [13].

O ST-II (*Stream Protocol version* II) é um protocolo de rede orientado a conexão, que opera no mesmo nível que o IP [14]. Ele foi desenvolvido para suportar a entrega de *streams* de dados em modo unicast ou multicast, para aplicações que necessitam de Qualidade de Serviço. O ST-II é parte da família de protocolos IP. Suas principais aplicações são o transporte de dados multimídia em tempo real e simulação distribuída. O ST-II trabalha com QoS através da reserva de largura de banda. A reserva de largura de banda, juntamente com mecanismos de acesso e escalonamento de pacotes garante a QoS desejada. O ST-II é composto de dois protocolos: o ST (*Stream Protocol*), para entrega de dados, e o SCMP (*Stream Control Message Protocol*), para o controle das funções do protocolo. As mensagens SCMP são transferidas dentro de pacotes ST [14].

O ST-II é baseado em um modelo de comunicação em dois passos. Primeiramente, os canais de tempo real são criados, na fase de configuração dos *streams*. Nesta fase são selecionadas as rotas, e os recursos são reservados. Na segunda fase, de transferência de dados, os canais de tempo real são utilizados para transferir dados com requisitos de QoS. Esta divisão é feita porque, na primeira fase, não é necessário garantir a QoS na transmissão [14].

O ST-II foi criado para ser usado com aplicações multimídia. Ele usa o mesmo tipo de endereços e o mesmo esquema de resolução que o IP. De fato, a diferença entre pacotes IP e pacotes ST-II é feita através do número da versão do protocolo: versão cinco para ST-II, e versões quatro ou seis para o IP.

Como o ST-II é um protocolo de rede, há vários protocolos de transporte construídos acima dele. Alguns exemplos são o RTP, comentado anteriormente, o PVP (*Packet Video Protocol*), o NVP (*Network Voice Protocol*), e o HeiTP (*Heidelberg Transport Protocol*) [10].

4 O Protocolo XTP

O XTP (*Xpress Transport Protocol*) é um protocolo de transporte que suporta aplicações convencionais e aplicações de tempo real. Ele foi desenvolvido pelo XTP Forum, um consórcio formado por empresas, universidades e centros de pesquisa. O XTP foi desenvolvido para substituir o TCP como protocolo de transporte, complementando-o em vários quesitos. Suas principais características são [15]:

Separação entre paradigma e política Os mecanismos de controle do XTP são ortogonais, isto é, pode-se controlar cada um deles separadamente, sem afetar os demais. Por exemplo, o controle do paradigma de comunicação (datagrama, circuito virtual, transação), é separado da política de controle de erros (detecção, detecção e correção), permitindo uma grande flexibilidade na adaptação do protocolo. Pode-se habilitar ou desabilitar cada aspecto do XTP.

Separação entre controle de taxa e de fluxo O controle de fluxo opera fim-a-fim, controlando os buffers de comunicação. O controle de taxa opera entre produtor e consumidor. No XTP, há mecanismos para controlar fluxo e taxa independentemente.

Controle explícito de *multicast* confiável Todos os mecanismos do XTP disponíveis para comunicação *unicast* estão também disponíveis em modo *multicast*. Isto permite controlar os mecanismos do protocolo sem se preocupar com o número de parceiros na comunicação.

Independência do serviço de entrega de dados Para que o XTP funcione, somente se faz necessário que a camada de rede proporcione a segmentação e a entrega de dados entre hosts XTP. Isto é possível através de virtualmente qualquer protocolo de rede, ou ainda de protocolos de enlace. Há implementações do XTP funcionando acima de IP, UDP, AAL5 (implementado neste trabalho), e outros. O XTP possui um esquema de formato de endereçamento flexível, o que facilita seu uso em diversos protocolos de rede.

O XTP possui outras facilidades, dentre as quais estão: [15]

- Modo de conexão rápida para circuitos virtuais;
- Busca de endereços baseada em chave;
- Escalonamento e priorização de mensagens;
- Suporte a protocolos encapsulados e de convergência;
- Retransmissão e "*acknowledgement*" seletivos;
- Frames fixos e alinhados em 64 bits;
- Identificadores de conexão e de seqüência de 64 bits;
- Negociação de tráfego.

A seguir são apresentadas as características do XTP mais utilizadas neste trabalho.

4.1 Controle de Fluxo

O controle de fluxo no XTP é feito através de um esquema de janelas deslizantes (*sliding window*). As janelas são controladas pelo número de seqüência atribuído ao *stream* de dados. Um número de seqüência é dado a cada byte enviado. Há dois campos nos pacotes de controle que regulam o controle de fluxo. O valor do campo *alloc* indica o número máximo de seqüência permitido, isto é, a borda superior da janela. O campo *rseq* é setado com o valor imediatamente superior ao último byte recebido continuamente, o que corresponde à borda inferior da janela.

Há também dois flags que regulam o fluxo. O flag RES indica que o receptor deve indicar no campo *alloc* somente o tamanho do buffer que o usuário alocou para aquela associação. Isto é chamado de modo de reserva (*reservation mode*). O flag NOFLOW indica ao receptor que o transmissor não quer ser limitado pelo controle de fluxo, que é desabilitado na direção de transmissão. Caso o receptor não concorde com essa política, ele pode rejeitar a conexão enviando um pacote DIAG.

4.2 Controle de Erros

O controle de erros no XTP é feito através da troca de informações sobre dados perdidos ou danificados. Todo pacote é verificado através de *checksum*, e os dados perdidos são detectados e recuperados usando mecanismos de *acknowledgement* e retransmissão. A perda de pacotes de controle é detectada por *timeouts*, e a notificação de outros erros, como pacotes não esperados, é feita através de pacotes DIAG.

O *checksum* de pacotes XTP pode ser feito no pacote inteiro ou somente no cabeçalho, caso tenha sido enviado o flag NOCHECK. O algoritmo de *checksum* é o mesmo feito pelo IP, que é a soma do complemento-de-um dos pares de octetos (16 bits). Um receptor detecta pacotes que estejam faltando através de intervalos nos números de seqüência do *stream*. O receptor mantém o número de seqüência do primeiro byte que não chegou. Caso não haja "buracos" na seqüência, um pacote CNTL é enviado. Caso contrário, um pacote ECNTL é enviado.

Há duas formas de recuperar os dados perdidos: retransmitindo-se toda a seqüência entre o valor de *rseq* e o maior número de seqüência (*go-back-n*), ou transmitindo-se os intervalos não recebidos (*selective acknowledgement*). A técnica seletiva é mais eficaz em redes lentas e com alta taxa de erros, enquanto que *go-back-n* é mais eficaz em redes rápidas e com baixa taxa de erros, que é o caso deste trabalho. As notificações de erro

podem ser enviadas quando requisitadas, ou sempre que for detectado um "buraco" na seqüência, o que é chamado de FASTNAK (*fast negative acknowledgement*).

4.3 Controle de Taxa

O controle de taxa governa o relacionamento produtor-consumidor entre hosts XTP [15]. O controle de taxa está relacionado com a velocidade com que os dados são processados ou consumidos no receptor. Esse dado é informado ao transmissor, que pode assim controlar a emissão dos dados. A taxa negociada em uma direção pode ser diferente daquela negociada para a direção oposta.

No XTP o controle de taxa é feito através de uma especificação de tráfego, que é negociada durante o estabelecimento da conexão. Essa especificação contém três valores: a taxa, em bytes por segundo, o tamanho das rajadas (*bursts*), também em bytes por segundo, e o maior tamanho de MTU a ser usado na conexão. A negociação desses valores faz parte do controle de tráfego, explicado a seguir.

4.4 Controle de Tráfego

O controle de tráfego é feito através de uma especificação de tráfego. Uma especificação de tráfego é um contrato entre as aplicações que determina a forma do tráfego de uma conexão [15]. O XTP possibilita que vários formatos de especificação de tráfego sejam criados, definindo somente a negociação deles. A negociação é feita na forma de um *two-way handshake*.

O iniciador da negociação envia um pacote contendo os valores de tráfego sugeridos. O receptor pode então tomar três ações: aceitar a especificação de tráfego, aceitar com modificações, ou rejeitar a especificação. Nos dois primeiros casos é retornado um pacote TCNTL, e no caso de rejeição, um pacote DIAG. O iniciador, ao receber a resposta, pode concluir a negociação com sucesso, caso o receptor tenha concordado com a especificação, ou caso o iniciador concorde com as modificações. O iniciador deve encerrar a conexão, caso o receptor tenha rejeitado a especificação.

O protocolo XTP inclui ainda a renegociação do tráfego durante a conexão, embora essa característica não tenha sido implementada no SandiaXTP, que foi a implementação do XTP utilizada neste trabalho [16]. Todas as outras características de tráfego estão implementadas no SandiaXTP.

5 Um Modelo de QoS Multinível

O tema "Qualidade de Serviço" é relativamente novo. A maioria da literatura do assunto aborda QoS em um contexto específico, como o de aplicações multimídia, ou ainda em um só nível. Nosso objetivo é definir um modelo genérico, de múltiplos níveis e baseado em padrões de computação distribuída, mais especificamente, no modelo ODP.

O primeiro problema a ser resolvido no modelo proposto é a definição dos requisitos e métricas de QoS. Neste contexto, requisitos são as características de comportamento que serão avaliadas, e as métricas são os parâmetros utilizados para avaliar os requisitos. Algumas definições de requisitos e métricas existem na literatura, mas a maioria é dirigida a uma classe específica de aplicações, geralmente as aplicações multimídia [6,9,10,11].

São listados a seguir os requisitos e métricas definidos para o modelo proposto. Cada classe de aplicações somente necessitará de um subconjunto dos requisitos definidos.

Sincronismo

Controla o grau de sincronismo necessário à comunicação. Pode ser síncrono, assíncrono ou isócrono.

Interatividade

Controla aspectos específicos da interatividade na comunicação. As métricas são: *throughput*, delay, jitter e a MTU.

Capacidade

Controla algumas características do ambiente computacional, como: velocidade agregada de CPU (MIPS, MFLOPS), capacidade de disco e o tamanho da RAM agregada.

Prioridade

Especifica a prioridade da aplicação em relação às demais. No nosso modelo, somente haverão os valores "alta prioridade" e "baixa prioridade", por serem estes os valores suportados pela rede ATM.

Controle de Erros

Especifica as ações a serem tomadas na presença de erros, que podem ser: detecção, detecção e correção, ou ainda características específicas do protocolo de transporte, como veremos na próxima seção.

Tolerância a Falhas

Especifica os mecanismos de tolerância a falhas presentes no ambiente computacional. Indica a existência de redundância de host, linha de comunicação, CPU, RAM e fonte elétrica, além da classificação RAID (*Redundant Array of Independent Disks*) do sistema de discos.

Confidencialidade

Define o nível de confidencialidade usado na comunicação. Usamos para isto a classificação NSA (A1, B1, B2, etc).

Autenticidade

Define o nível de autenticidade usado na comunicação. Usamos a classificação NSA.

Integridade

Define o nível de integridade usado na comunicação. Usamos a classificação NSA.

Disponibilidade

Especifica o grau de disponibilidade do ambiente computacional, medido pelas métricas MTBF (*Mean Time Between Failures*) e MTTR (*Mean Time to Repair*).

Custo

Especifica uma unidade de custo dos recursos computacionais, como: custo por minuto de CPU, por megabyte armazenado, por página impressa e por octeto transmitido.

O próximo passo do modelo proposto é especificar um processo de negociação destes requisitos com as aplicações ODP. Alguns destes requisitos estão diretamente ligados à infra-estrutura de comunicação, e têm características dinâmicas. Outros são de mais alto nível, dependem de fatores não relacionados à rede, e são relativamente estáticos [7]. Portanto, tornou-se necessário dividir a negociação em duas etapas.

Inicialmente, os requisitos de alto nível são negociados com o Trader ODP, e na etapa seguinte, os requisitos de rede são negociados através de um Objeto de Negociação de QoS (ONQoS). Os requisitos negociados com o Trader especificam características do ambiente computacional, enquanto aqueles negociados com o ONQoS especificam as características desejadas para o canal de comunicação. Mais especificamente, os requisitos de sincronismo, interatividade, controle de erros e prioridade são negociados pelo Objeto de Negociação de QoS, e os outros através do Trader.

De acordo com o tipo da aplicação, somente uma das etapas pode ser necessária. Aplicações que só estão preocupadas com requisitos de alto nível usam somente a primeira etapa. Aplicações que só precisam de garantias da rede, utilizam somente a segunda fase. Aplicações mais complexas podem precisar de ambas. Por exemplo, se é necessária uma comunicação de voz autenticada, as métricas de autenticação são negociadas com o Trader, e os requisitos para a transmissão de voz são negociados com o ONQoS.

A negociação com o Trader usa as interfaces de exportação e importação citadas anteriormente. Essas interfaces são bastante flexíveis, e permitem realizar pesquisas por vários critérios. A negociação com o ONQoS utiliza uma interface própria, que será descrita na próxima seção. A Figura 5 mostra o modelo proposto. Na figura, os círculos indicam os objetos implementados neste trabalho, os quadrados indicam elementos já existentes. As linhas pontilhadas indicam chamadas ORB, e as linhas contínuas indicam chamadas locais.

No RM-ODP, os BEOs (*Basic Engineering Objects*) são as aplicações usuárias do ambiente. Um BEO pode ter o papel de cliente e/ou servidor. Um BEO atua como cliente quando está invocando um serviço, e atua como servidor quando está respondendo a pedidos de invocação. Detalharemos a seguir o processo de negociação.

Inicialmente o BEO servidor cria seus serviços e registra seus requisitos de QoS de alto nível com o Trader. O Repositório de Implementação está no modelo somente para indicar que os requisitos de QoS podem ser armazenados junto com a implementação do Objeto Servidor. O BEO servidor também registra seus requisitos de QoS de rede com o ONQoS. O ONQoS tem acesso a um repositório, para que os registros de QoS possam tornar-se persistentes.

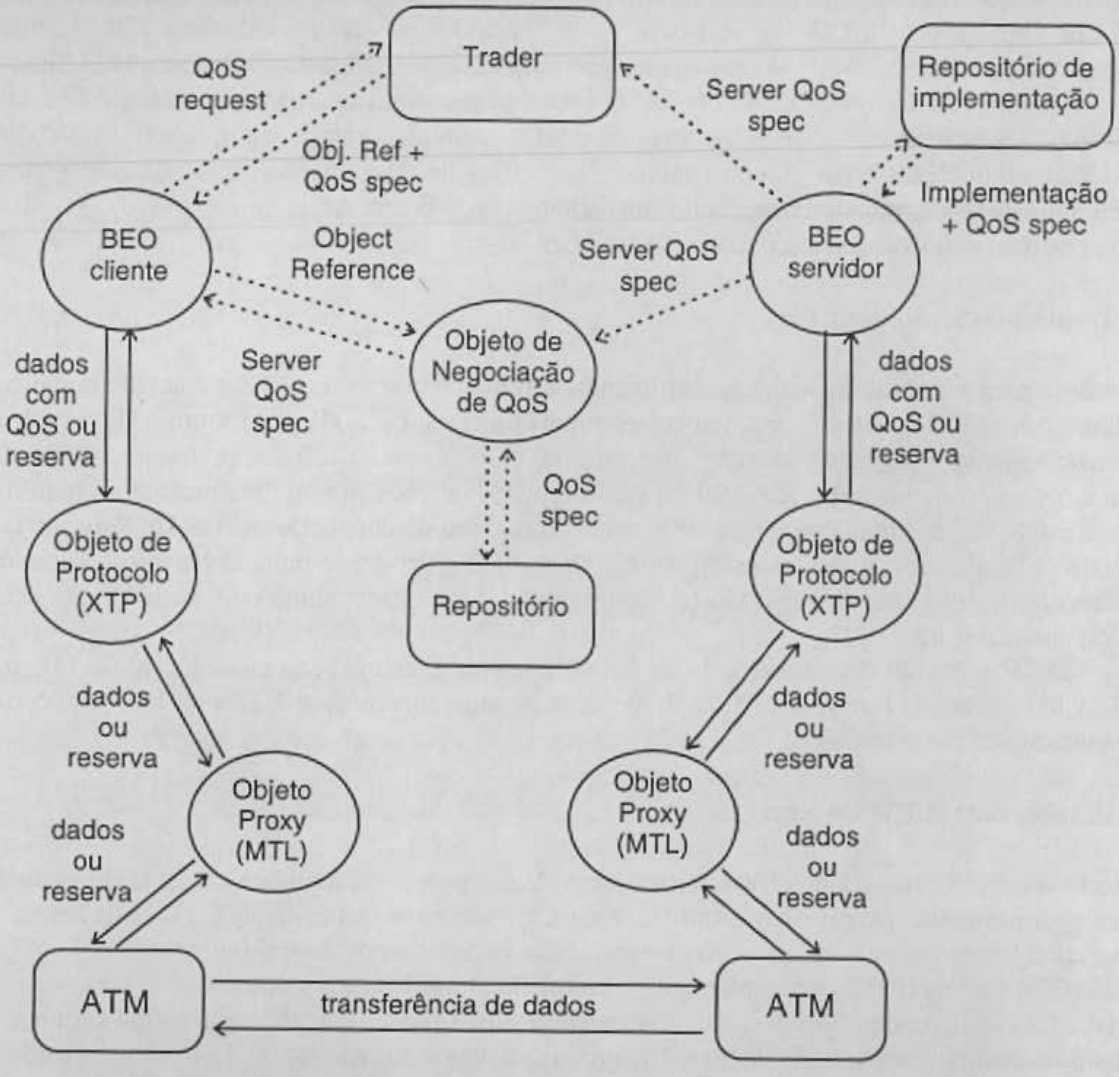

Figura 5: O modelo proposto

Quando um BEO cliente deseja algum serviço, ele tem duas alternativas. Caso ele conheça a referência do Objeto Servidor, ele a submete ao Trader, que devolve a especificação de QoS do servidor. Caso o BEO cliente queira qualquer servidor que se encaixe em seus requisitos, ele envia os requisitos desejados, e o Trader retorna uma referência de objeto. Essa é a primeira etapa da negociação.

Na segunda etapa, quando o BEO cliente já tem a referência do BEO servidor, ele a envia ao ONQoS. A partir da referência, o ONQoS obtém os requisitos de QoS do BEO servidor e os entrega ao BEO cliente. Esses requisitos são passados ao Objeto de Protocolo na forma de um pedido de reserva de recursos. Esse pedido é atendido em parte pelo Objeto de Protocolo e em parte pelo Objeto Proxy. Um *handler* é retornado ao BEO cliente, que o utiliza para transferir os dados. O BEO servidor também pode fazer uma reserva de QoS, já que a QoS só é reservada em uma direção.

O Objeto de Protocolo tem por função ser um protocolo de transporte com funções de tempo real e requisitos de QoS. O Objeto Proxy tem duas funções. A primeira é ser um protocolo de rede genérico baseado em AAL5. A interface do Objeto Proxy é usada pelo Objeto de Protocolo para implementar a camada de transporte. A segunda função é mapear o pedido de reserva de recursos nos parâmetros de tráfego das redes ATM. As diversas estruturas e interfaces mostradas no modelo serão detalhadas na próxima seção.

6 Aspectos de Implementação

6.1 Ambiente de implementação

Conforme dissemos anteriormente, o modelo proposto foi desenvolvido tendo-se em mente sua implementação em uma rede ATM, por causa de suas características de alta velocidade e da existência de mecanismos de controle de tráfego.

A implementação foi realizada no Laboratório de Redes de Alta Velocidade do DCC/UFMG. O ambiente é composto de um comutador CISCO Hyperswitch A100, com 15 portas de 155Mbps (OC-3), usando fibra multimodo e par trançado categoria 5. Ao comutador estão ligadas duas SPARC Ultra 1, de 145Mhz, com 64 Mb de RAM, além de uma SPARC 5 com 32Mb de RAM. Todas as máquinas possuem placas SunATM 2.0 e sistema operacional Solaris 2.5.1. As duas SPARC Ultra foram usadas na implementação, já que a transferência de dados em alta velocidade exige CPUs e barramentos rápidos. Em testes de desempenho, notamos que o barramento da SPARC 5 não foi capaz de transferir os dados na velocidade máxima da interface ATM. As SPARC Ultra conseguiram aproximar-se da velocidade máxima da interface.

6.2 Implementação do XTP

Como base para nossa implementação, utilizamos a implementação do XTP conhecida como SandiaXTP, do Sandia Laboratories [16,17]. Esta é a implementação mais utilizada do XTP, por ter um código bem estruturado e por ser constantemente atualizada, já sendo compatível com a versão 4.0 do protocolo. O SandiaXTP foi implementado em C++, o que o torna mais simples de adaptar. Notamos porém dois problemas com o código. O primeiro deles é o fato de ser um pouco lento, em virtude do número de chamadas de função. Partindo do mais alto nível, até 18 chamadas de função são necessárias para chegar até o driver de rede. O outro problema é a falta de *threads*. Por executar em um único processo, o protocolo pode ter seu desempenho comprometido quando há vários clientes fazendo chamadas a ele.

O SandiaXTP é baseado em uma biblioteca de classes de comunicação chamada MTL (*Meta Transport Library*) [18]. A MTL trata da interface com os diversos protocolos nos quais o XTP está baseado, como IP, UDP ou AAL5 (implementado neste trabalho).

6.3 A interface ATM da Sun

A interface com as placas SunATM é feita através de um padrão de interface da Sun, chamado STREAMS [19,20]. Nesse padrão, temos um driver comum (/dev/sa0), que suporta chamadas IOCTL. Através das chamadas IOCTL é possível enviar e receber dados, além de especificar os parâmetros do tráfego. O driver é compatível com a especificação ATM Forum UNI 3.0, e implementa o protocolo Q.2931 para sinalização.

O driver tem dois modos de operação, *raw mode* e DLPI *mode*, que indicam o tipo de encapsulamento usado. No modo *raw* um único bloco de dados é enviado, contendo o número do VCI (4 bytes) seguido dos dados. A multiplexação no receptor é feita com base no VCI. No modo *DLPI (Data Link Provider Interface)* são enviados dois blocos, o primeiro contendo o tipo de mensagem DLPI, e o segundo contendo um cabeçalho LLC (*Logical Link Control*) seguido dos dados. A multiplexação é feita com base no cabeçalho LLC [20]. Por não necessitarmos da multiplexação de LLC, optamos por utilizar o modo *raw*, definindo-se um VCI padrão para ser usado no XTP (VCI 120).

Os parâmetros de tráfego são: o tipo de conexão (CBR ou VBR), a unidade de alocação de banda (64k ou 1Mbps), a taxa de pico, a taxa média, o número de bytes em modo *burst* e a prioridade. Pode-se especificar também se a alocação é para um VCI específico ou para qualquer VCI que usar a interface. Porém, no driver das estações Ultra há um problema com a alocação por VCI, ainda não corrigido na versão 2.0 do driver. Para contornar este problema, elaboramos um esquema com várias aberturas simultâneas do driver, que fazem com que, mesmo usando um só VCI, o tráfego dos diversos clientes não seja "misturado". Há ainda outros problemas na versão 2.0 do driver, ainda não corrigidos pela Sun. Através de um desses problemas, um usuário comum pode "rebootar" qualquer estação que tenha a placa SunATM. A versão do driver utilizada é baseada em PVC (*Permanent Virtual Calls*), embora já haja uma atualização para torná-lo compatível com SVC (*Switched Virtual Calls*) [19,20].

O maior throughput possível no nível STREAMS é de 134 Mbps. Dos 155 Mbps, 2 Mbps são reservados para controle, e o restante é perdido com cabeçalhos de células e da AAL5. Desses 134 Mbps, a MTLQoS reserva 2 Mbps para controle do protocolo, restando 132 Mbps para os usuários.

6.4 Portando o XTP para ATM

Conforme dissemos anteriormente, o SandiaXTP é baseado na MTL, que trata dos protocolos de rede nos quais o XTP está baseado. Para portar o XTP para ATM, reescrevemos a MTL para torná-la compatível com a AAL5. Chamaremos essa implementação modificada de MTLQoS.

A primeira decisão de projeto foi estabelecer o tamanho da unidade de transferência do protocolo (MTU). Através da interface STREAMS é possível alocar um buffer de transferência de até 64 kbytes, mas a MTL originalmente limita o tamanho da MTU em 8 kbytes. A Figura 6 mostra a relação entre o tamanho da MTU e o *throughput* no nível do *driver*. Conforme vemos no gráfico, acima de 4096 bytes praticamente não há mudança no

throughput, que se estabiliza em aproximadamente 128 Mbps. Porém, no nível do XTP, a MTU guarda uma relação estreita com outros parâmetros, conforme veremos na análise de desempenho. Analisando esses dados em conjunto, definimos uma MTU variável de 1 a 32kbytes, com valor *default* de 8kbytes.

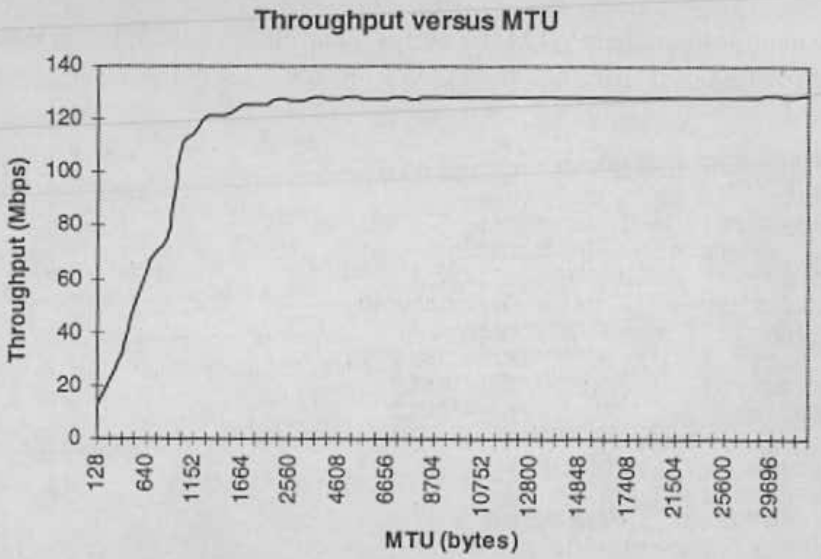

Figura 6 : Gráfico de throughput x MTU

Outra decisão foi quanto ao endereçamento. O XTP possui um esquema de endereçamento flexível, que permite definir novos formatos. Desta forma, seria possível utilizar os endereços NSAP (*Network Service Access Point*) (ISO 8348 e ITU X.213) implementados no driver, de 20 bytes. Por outro lado, se usarmos endereços IP ganharemos em portabilidade e facilidade. Por esse motivo, decidimos usar endereços IP. Outra questão analisada foi como mapear endereços IP em números de VCI. Para isto, criamos um esquema de ARP (*Address Resolution Protocol*) manual. Em cada estação ATM existe um arquivo de configuração que lista, em cada linha, o endereço IP seguido do VCI correspondente. Na inicialização do protocolo esta lista é colocada em memória, e usada sempre que se faz um *bind* no XTP.

6.5 Introduzindo QoS no XTP

Todo o mecanismo de QoS introduzido no XTP foi feito através de polimorfismo. Isto é, foram acrescentadas implementações de métodos que incluem parâmetros de QoS. Desta forma, a introdução de parâmetros de QoS não modificou a operação do XTP.

As aplicações já desenvolvidas para o XTP não precisam ser modificadas. Cada aplicação opta por utilizar ou não os parâmetros de QoS. Mostramos a seguir a estrutura definida para controlar a QoS vinda da rede.

```
typedef struct {
   int handle;          // handle para transmissão
    byte8 type;         // CBRxMEGA, CBRx64, VBRxMEGA
                        // VBRx64
   short16 peak;        // throughput de pico
   short16 average;     // throughput medio
   short16 burst;       // bytes em modo burst
   byte8 priority;      // prioridade (alta ou baixa)
} qos_type;
```

O parâmetro "type" indica se a transmissão será em modo CBR ou VBR, e se a unidade de alocação será de 64kbytes ou em megabytes. Os demais parâmetros são auto-explicativos. O *bind* é feito passando-se os seguintes parâmetros:

```
int bind(address_segment* addr,qos_type* qos_spec);
```

Se efetuado com sucesso, o *bind* retorna na estrutura um *handler*, a ser usado para o envio de dados. A função *send* possui quatro formas, onde duas recebem o *handler* como parâmetro. Mostramos uma delas a seguir.

```
int send(int block, short16* options, int shandle);
```

A função *release* libera a QoS alocada, e também encerra a conexão. Ela tem o seguinte protótipo:

```
int release(int rhandle);
```

Além desses parâmetros de QoS, garantidos pela rede ATM, há outros, controlados pelo XTP. A estrutura que propusemos e implementamos para controlar todos os parâmetros de QoS é mostrada a seguir.

```
struct qos_param {
  unsigned char    InSyncType; // CBR, VBR
  unsigned short   InPeakRate; // Taxa de pico
  unsigned short   InAverageRate; // Taxa media
  unsigned short   InBurst; // Tamanho do burst
  unsigned char    InPriority; // HIGH_PRI ou LOW_PRI
  unsigned char    InTypeOfService; // Tipo de conexao
  unsigned char    InErrorControl; GO-BACK, FASTNAK
  unsigned char    InFlowControl;  // DONTUSE, USEIT
  unsigned char    InCheckControl; // DONTUSE, USEIT
  unsigned char    InRateControl;  // DONTUSE, USEIT
  unsigned short   InMaxData; // MTU : 1-32K
  unsigned char    OutSyncType; // CBR, VBR
  unsigned short   OutPeakRate; // Taxa de pico
  unsigned short   OutAverageRate; // Taxa media
  unsigned short   OutBurst; // Tamanho do burst
  unsigned char    OutPriority; // HIGH_PRI ou LOW_PRI
  unsigned char    OutTypeOfService; // Tipo de conexao
  unsigned char    OutErrorControl; GO-BACK, FASTNAK
  unsigned char    OutFlowControl;  // DONTUSE, USEIT
  unsigned char    OutCheckControl; // DONTUSE, USEIT
  unsigned short   OutMaxData; // MTU: 1-32K
};
```

Os primeiros cinco parâmetros correspondem aos da estrutura anterior. O parâmetro InTypeOfService controla o tipo de conexão: *connection oriented, transaction, unacknowledged datagram, acknowledged datagram, isochronous* ou *bulk*. A especificação do XTP descreve em detalhes cada um desses modos [15]. O parâmetro InErrorControl especifica os mecanismos de controle de erro usados: *go-back-n* (o default é usar *selective retransmission*) e/ou *fast negative acknowledgement*. Os parâmetros InFlowControl, InCheckControl e InRateControl regulam o uso de controle de fluxo, checksum nos dados e controle de taxa, respectivamente. O parâmetro InMaxData especifica o tamanho da MTU a utilizar. Os demais parâmetros são os mesmos, válidos para o *stream* de sentido contrário.

Esses parâmetros são passados ao XTP através de duas funções: *configure_session* e *configure_traffic*. A primeira controla os parâmetros de sessão, que são estabelecidos antes do *bind*, e a segunda controla os parâmetros de tráfego, que são estabelecidos depois do *bind* e antes da transferência dos dados.

6.6 O Objeto de Negociação de QoS

O Objeto de Negociação de QoS (ONQoS) foi implementado usando-se RPC (*Remote Procedure Call*), com uma interface compatível com a CORBA-IDL. O ONQoS possui uma interface simples, com funções para incluir, modificar e retirar serviços. A chave de busca é sempre a referência do objeto.

Um ONQoS deve existir em cada host servidor. O objeto servidor cadastra seus serviços no ONQoS, que podem ser modificados, conforme são criadas novas versões. As tabelas de serviços são mantidas em memória. Nenhuma função de persistência foi criada, embora sua implementação seja simples. A interface do ONQoS é feita através das seguintes funções:

```
int *register_service(struct srecord *ptr);
int *drop_service(unsigned long *objref);
int *change_service(struct srecord *ptr);
struct srecord *give_qos(unsigned long *objref);

struct srecord {
   unsigned long    ObjRef;
   unsigned char    deleted;
   qos_param        rep_qos;
}
```

As funções incluem, removem, modificam e pesquisam serviços, respectivamente. É mostrada também a estrutura "srecord". O campo ObjRef guarda a referência do objeto, que varia de formato de acordo com a plataforma ORB utilizada. Nós a definimos como um "long" sem sinal (64 bits). O campo "deleted" ajuda a remover os serviços na memória. A estrutura "qos_param", como vimos anteriormente, guarda os parâmetros de QoS.

6.7 Análise de Desempenho

Nesta seção analisaremos o desempenho do XTP com QoS (XTPQoS) em uma rede ATM. Conforme dissemos anteriormente, o protocolo XTP é bastante complexo. A implementação SandiaXTP é composta de mais de 20.000 linhas de código C++, e com muitas chamadas de função. O fato do protocolo executar em um único processo também contribui para os problemas. Apesar disto, como veremos adiante, o desempenho mostrou-se satisfatório, principalmente por causa das estações de alta velocidade utilizadas. Comparando-se os resultados com os requisitos para transmissão de aplicações multimídia, como vídeo MPEG, vemos que é perfeitamente possível sua utilização. Veremos a seguir alguns gráficos de desempenho.

A Figura 7 mostra a relação entre a banda alocada e o *throughput* efetivamente obtido no XTPQoS. São mostrados uma conexão em modo não-bloqueante, onde o envio de dados é assíncrono em relação às confirmações de recepção, e uma conexão bloqueante, onde cada pacote só é enviado depois do pacote anterior ter sido confirmado. Utilizamos no exemplo pacotes de 32 kbytes Vemos que no exemplo não-bloqueante o *overhead* do protocolo é mínimo, e o *throughput* cresce linearmente até um maximo de aproximadamente 110 Mbps. No caso bloqueante, o limitante é a espera pelas confirmações, que restringe o *throughput* a aproximadamente 40 Mbps.

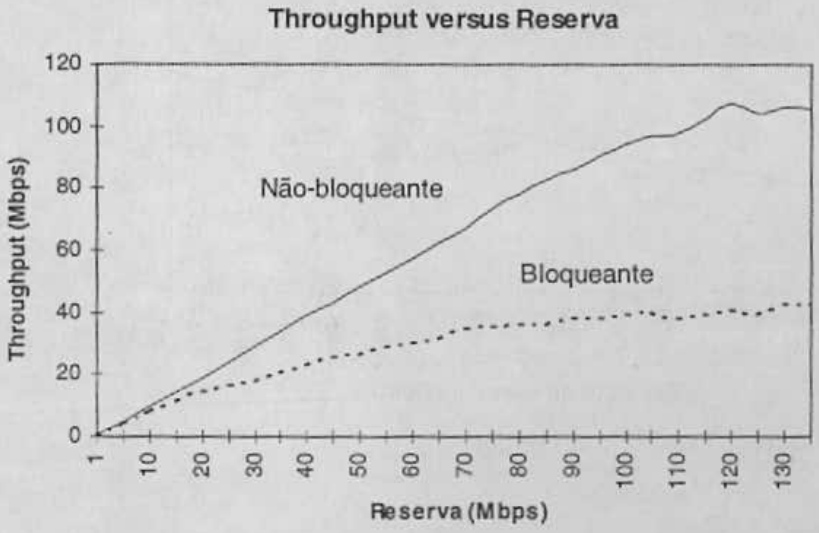

Figura 7: Reserva x Throughput

Na Figura 8 vemos a relação entre o *throughput* e a MTU. Vemos que, quanto maior a MTU, maior é o *throughput*. Com uma MTU pequena, o *overhead* de troca de mensagens e de processamento do protocolo passa a ser mais significativo, e temos um menor *throughput*.

A Figura 9 mostra a relação entre o delay e o tamanho dos pacotes no modo bloqueante. Aqui vemos que o delay cresce com o tamanho dos pacotes. O delay máximo apresenta uma grande variação, que atribuimos ao escalonamento do processador durante a transmissão. Porém, o delay médio cresce linearmente, chegando ao máximo de 5 ms. No modo não-bloquente o delay médio é ainda menor, não passando de 2 ms.

Na Figura 10 vemos a relação entre o jitter e o tamanho do pacote em modo bloqueante. Podemos ver que, da mesma forma que o delay (e pela mesma causa), a variação do jitter máximo é bem grande, porém o valor médio é baixo, menos de 0.2 ms. A variação pode ser facilmente compensada usando-se buffers de tamanho compatível.

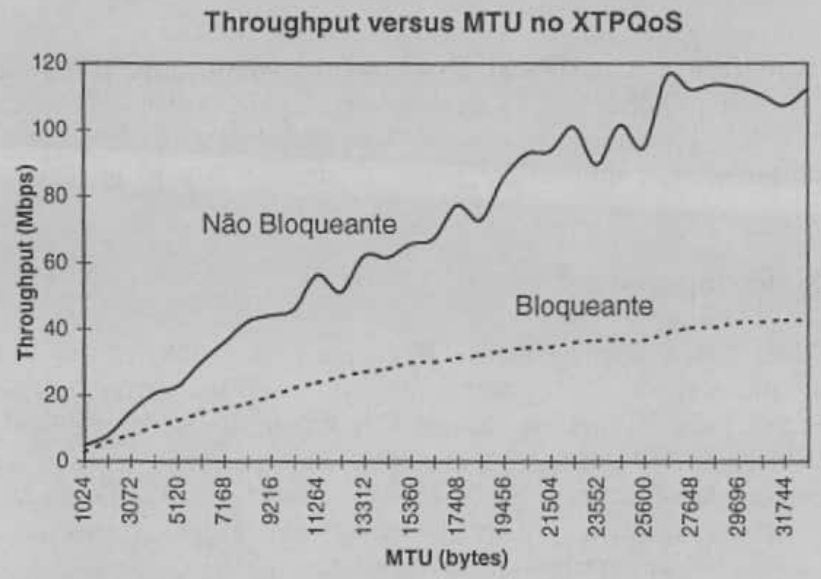

Figura 8: Throughput x MTU

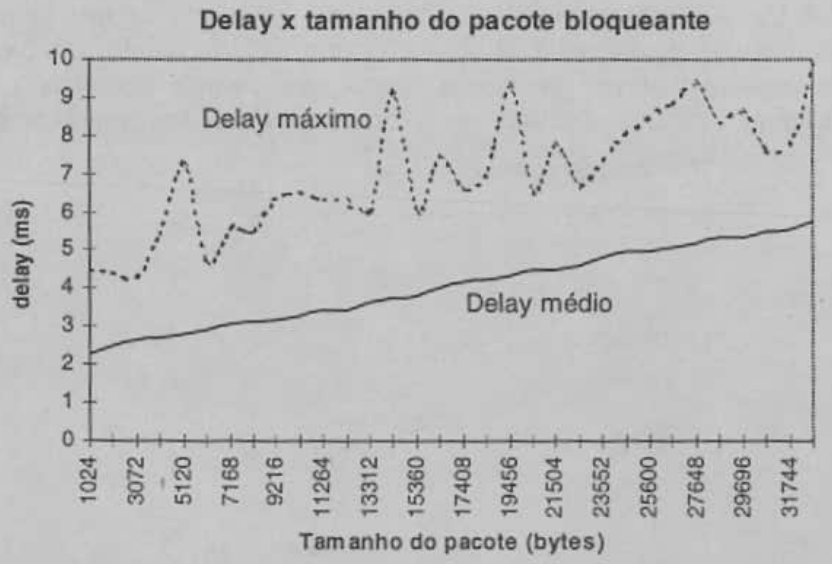

Figura 9: Delay x Tamanho do pacote

7 Conclusão

Neste trabalho apresentamos um modelo de Qualidade de Serviço em múltiplos níveis, baseado em redes ATM, no modelo ODP e na arquitetura CORBA. Para validar o modelo, implementamos um ambiente integrado de QoS, consistindo de um protocolo de rede baseado em ALL5 (MTLQoS), um protocolo de transporte de tempo real com requisitos de QoS (XTPQoS), e um objeto de negociação distribuída de QoS (ONQoS), baseado na arquitetura CORBA.

Constatamos que, tanto a arquitetura CORBA quanto o modelo ODP, ainda são muito genéricos ao tratar de Qualidade de Serviço. Com a evolução desses modelos, certamente teremos descrições mais precisas, que, por exemplo, definam os requisitos e métricas relevantes em um ambiente de computação distribuída. Neste trabalho propusemos um conjunto desses requisitos e métricas.

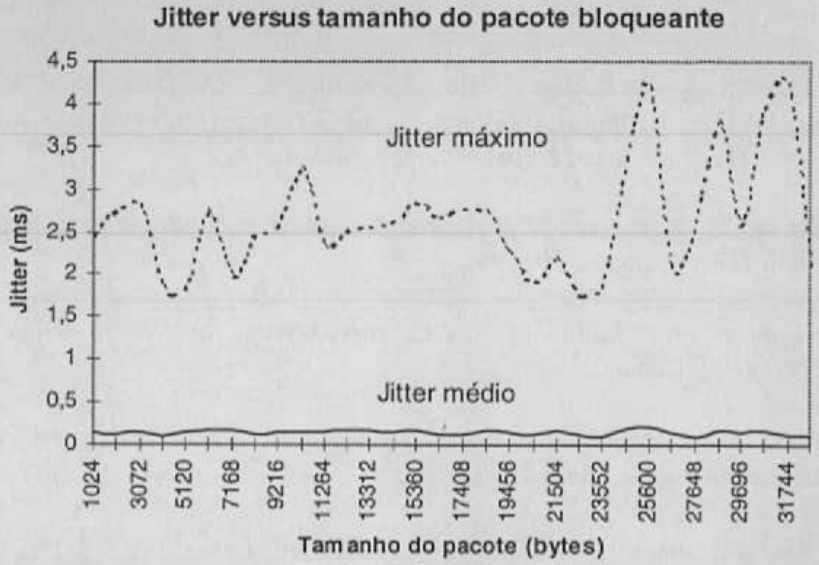

Figura 10: Jitter x Tamanho do pacote

Através da implementação pudemos visualizar os desafios de integrar redes ATM, protocolos de tempo real e requisitos de QoS. Os protocolos de tempo real, como o XTP, tendem a ser mais complexos que os protocolos comuns, e por isto, exigem maior processamento nas estações. Por outro lado, não se pode gastar muito tempo no processamento, já que a rede ATM é capaz de despejar milhares de pacotes por segundo, que devem ser processados e entregues dentro dos limites de QoS especificados. Isto exige protocolos altamente otimizados e CPUs rápidas.

Pela análise de desempenho pudemos verificar a viabilidade de transmitir tráfego multimídia usando XTP sobre ATM, e as vantagens de poder especificar e negociar parâmetros de QoS desde o nível de aplicação. Há uma forte tendência nos protocolos modernos em incluir parâmetros de QoS, e também há um grande crescimento no uso de redes ATM. Portanto, é bastante provável que vejamos várias implementações nesta linha nos próximos anos.

Agradecimentos

Os autores agradecem ao professor José Marcos Silva Nogueira, do DCC/UFMG, por nos receber no Laboratório de Redes de Alta Velocidade durante a realização deste projeto. Sem sua ajuda, nada disto teria sido possível.

Os autores agradecem também o apoio financeiro da FAPESP, do CNPq e do Tribunal Superior do Trabalho.

Referências

[1] ISO/IEC JTC1/S C21, *Basic Reference Model ODP - Part 1: Overview and Guide to Use*, junho de1995.

[2] ISO/IEC JTC1/S C21, *Basic Reference Model ODP - Part 2: Descriptive Model*, janeiro de 1995.

[3] ISO/IEC JTC1/S C21, *Basic Reference Model ODP - Part 3: Prescriptive Model*, janeiro de 1995.

[4] Object Management Group. *The Common Object Request Broker: Architecture and Specification revision 1.2*. OMG TC Document 93.12.29, dezembro de 1993.

[5] W.P.D.C. Loyolla; E.R.M. Madeira; M.J. MENDES; E. CARDOSO; M.F. MAGALHÃES. "*Multiware Platform: an Open Distributed Environment for Multimedia Cooperative Applications*". IEEE Computer Software & Applications Conference, Taipei, Taiwan, novembro de 1994.

[6] A. Vogel; V. Bochmann; R. Dssouli; J. Gecsei; A. Hafid; B. Kerherve. "*On QoS Negotiation in Distributed Multimedia Applications*". Technical Report 891, Université de Montréal, Montréal, Canada, 1993.

[7] F.H.S. Lima; E.R.M. Madeira. "*ODP-based QoS Specification for the Multiware Platform*". Proceedings of the IV IFIP International Workshop on Quality of Service, pp. 45-54. Paris, France, março de 1995.

[8] J. De Meer. "*Quality of Service in Open Distributed Processing*". IFIP International Conference on Open Distributed Processing, invited talk, Brisbane, Australia, fevereiro de 1995.

[9] A. Vogel; V. Bochmann; R. Dssouli; J. Gecsei; A. Hafid; B. Kerherve. "*On Distributed Multimedia Presentational Applications: Functional and Computational Architecture and QoS Negotiation*". Proceedings of the Fourth International Workshop on Protocols for High-speed Networks, Honolulu, 1994.

[10] A. Vogel; V. Bochmann; J. Gecsei; B. Kerherve. "*Distributed Multimedia Applications and Quality of Service - A Survey*". IEEE Multimedia, vol. 2, no. 2, pp. 10-19, 1995.

[11] A. Hafid; V. Bochmann. "*An approach to Quality of Service Management for Distributed Multimedia Applications*". Université de Montréal, Montréal, Canada.

[12] L.A.P. Lima junior; E.R.M. Madeira. "*A Model for a Federative Trader*". Open Distributed Processing: Experiences with Distributed Environment. Chapman & Hall, 1995.

[13] H. Schulzrinne; S. Casner; R. Frederick; V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Request for Comments 1889, janeiro de 1996.

[14] C. Topolcic. "*Internet Stream Protocol*". Request for Comments 1190, outubro de 1990.

[15] XTP Forum. *Xpress Transport Protocol Specification revision 4.0*, março de 1995.

[16] Sandia National Laboratories. *SandiaXTP Reference Manual*, setembro de 1995.

[17] Sandia National Laboratories. *SandiaXTP User's Guide*, outubro de 1995.

[18] Sandia National Laboratories. *Meta Transport Library User's Guide*, outubro de 1995.

[19] Sun Microsystems. *SunATM 155 Sbus Cards Manual*, maio de 1995.

[20] Sun Microsystems. *SunATM Sbus Adapters Manual*, janeiro de 1996.