Qualidade de Serviço a Nível de Aplicação

Marcos Tadeu von Lutzow Vidal

maio/97

Resumo

As aplicações multimídia com taxa de bits variável (VBR) possuem duas características notáveis: as unidades de dados do nível de aplicação são segmentadas gerando diversos pacotes que são enviados pela rede como uma explosão; para se garantir qualidade de serviço (QoS), como atraso e perda, para a aplicação, é necessário garantir limites de atraso e perda para as explosões de pacotes, como um todo, considerando-se o trajeto fim-a-fim, de aplicação a aplicação. Para isto, deve-se ter comutadores que permitam manipular requisitos de QoS para explosões de pacotes e, também, o sistema operacional das máquinas envolvidas no serviço multimídia devem manipular adequadamente os pacotes provenientes de aplicações multimídia que tenham requisitos de garantia de qualidade de serviço. Este trabalho apresenta diversas técnicas que permitem alcançar este objetivo.

1 Introdução

Com a evolução da multimídia, os desenvolvedores cada vez mais se preocupam com a qualidade, de maneira a manter uma apresentação multimídia suave e confortável. Em uma aplicação multimídia, um evento é normalmente codificado em uma unidade de dados da aplicação (ou protocolo de nível alto). Esta unidade de dados quase sempre é transmitida através de diversos pacotes, devido a necessidade de adequação ao limite imposto nas camadas inferiores, quanto ao tamanho máximo de unidades de dados (pacotes).

Um vídeo, por exemplo, tem cada uma de suas imagens codificadas em uma unidade de dados. Cada unidade de dados é segmentada para se adequar ao meio de transmissão, em termos de número de octetos por pacote, formando uma explosão de pacotes, ao ser transmitida pela rede.

Para o usuário, o mais importante é a suavidade da apresentação como um todo. Assim, para uma aplicação, é mais importante o atraso e/ou perda de uma unidade de dados da aplicação (ADU - *application data unit*) [1] do que dos pacotes individuais que formam este evento: deve-se tratar a explosão de pacotes como um todo [2]. Para se conseguir isto, não adianta garantir a qualidade de serviço a nível de pacotes, se não se garante a qualidade a nível de explosão de pacotes.

Assim, não basta garantir qualidade de serviço a nível de rede, ou seja, a nível de transmissão de pacotes. Esta garantia é necessária, mas não suficiente. é necessário também garantir a qualidade entre os serviços de rede e a aplicação. Para isto, deve-se ter uma arquitetura de *I/O* de rede que privilegie a alta eficiência, estando preparada para fazer tratamentos especiais a determinados fluxos de dados [3].

No caso de eventos armazenados, é necessário também fazer tratamentos especiais, de modo a não permitir que uma eventual maior solicitação ao sistema de disco venha a ocasionar atrasos na recuperação de dados referentes a apresentações multimídia.

Este trabalho apresenta diversas técnicas propostas, algumas implementações e resultados de pesquisas que visam adequar o sistema operacional de uma máquina a trabalhar com as novas exigências das aplicações multimídia.

2 Tratando explosões na rede

Diante da necessidade de se tratar os pacotes oriundos da segmentação de uma unidade de dados da aplicação como um todo, é usado o conceito de explosão [4]. Considerando-se uma aplicação que gere uma seqüência de unidades de dados endereçadas a um mesmo destino, cada ADU pode ser segmentada e carregada por um grupo de pacotes: uma explosão. O nome se deve ao fato de que, ao ser segmentada em pacotes, estes estão prontos para envio para a rede com um mínimo de processamento, fazendo com que estes pacotes tenham pouco tempo entre o envio de um e outro. Assim fica caracterizada uma explosão de pacotes pela rede, gerando uma taxa média geralmente bem mais alta do que a taxa média gerada pela aplicação, durante sua utilização.

As características do tráfego gerado por um vídeo codificado em MPEG, adequão-se perfeitamente ao conceito de explosão [5].

2.1 QoS para explosões

Os comutadores de pacotes usados em integração de serviços devem ser projetados para proverem garantia de QoS tanto para pacotes individuais quanto para explosões [6].

Uma classe de rede de comutadores de pacotes, chamada de rede Manipuladora de Explosões foi apresentada em [4]. Esta classe tem as seguintes características:

- para determinar a prioridade de serviço, entre os diversos pacotes, usa relógio virtual (VC) [7];
- garantia de atraso fim-a-fim e garantia de variação de atraso (*jitter*) para os fluxos de dados [8];
- escalonamento eficiente de tráfego de explosões, através da melhoria dos algoritmos, que exploraram as características do modelo de tráfego e garantia de atraso.
- garantias de *firewall*: se uma determinada fonte de tráfego passar a enviar tráfego acima do negociado, nenhum dos outros usuários do serviço será prejudicado; o único usuário a ser penalizado com perda de pacotes será a fonte defeituosa.

Os servidores que tem garantia de entrega com limite de atraso de pacotes que fazem parte de uma explosão podem ter uma eficiência muito maior [9, 10]. Esta eficiência é conseguida no algoritmo de escalonamento de prioridade de serviço (transmissão) pôde ser muito melhorada, ao se explorar a característica de poder derterminar a prioridade (executar o algoritmo) apenas uma vez para cada explosão, ao invés de uma vez para cada pacote.

A qualidade de serviço para explosões, a nível de rede, pode ser estudada para diversas fontes de tráfego. O fluxo gerado por cada uma destas fontes é dado pela seqüência de pacotes por ela gerados. Durante a fase de conexão, cada fonte de fluxo f deverá negociar, com o prestador do serviço, uma taxa de fluxo *r(f)* para a explosão (não é a taxa média do fluxo). Ao chegar a um comutador, um pacote p, de tamanho *l(p)*, é marcado pela sua hora de chegada ($A(p)$ - *Arrival time*), sempre maior que 0 (zero). A saída de p do comutador será marcada pela hora *L(p)*.

Cada fluxo de dados tem agregado a si um valor de relógio virtual (VC). Este valor é determinado pela soma de duas parcelas: a primeira é dada pelo máximo (o que for maior) entre a hora de chegada do pacote mais recente e o valor atual do relógio virtual do fluxo; a segunda parcela é dada pelo tempo de transmissão de *l(p)* bits a uma taxa *r(f)* bps. Este tempo não é o tempo de transmissão do pacote em questão, sendo calculado como se existisse um canal de capacidade *r(f)* bps alocado para transmitir unicamente o fluxo *f*. Este valor (VC) passa a ser a estampa de tempo *P(p)* do pacote. O valor de relógio virtual calculado é usado tanto para o fluxo como para o pacote. Assim, o valor do relógio virtual do fluxo (*P(f)*) é o relógio virtual do pacote (*P(p)*) que chegou mais recentemente.

A prioridade no escalonamento para serviço (transmissão) será daqueles pacotes com o menor valor de relógio virtual (*P(p)*), ou seja, o pacote com o menor *P(p)* será o primeiro a ser servido.

Um sistema (fila de entrada mais servidor) é dito ativo, se o sistema contém pelo menos um pacote esperando para ser servido (não está vazio) e, em um dado instante t, o valor de *P(f)*é maior do que t. Dada a capacidade de serviço do servidor, C bps, diz-se que este servidor está com sua capacidade excedida se a soma das taxas *r(f)*, para todo fluxo *f* ativo, for maior que C.

Se a capacidade de servidor não estiver excedida, *L(p)* deverá ser menor ou igual a *P(p)* somado ao tempo de transmissão do pacote de maior tamanho permitido, pelo link de saída (capacacidade C) [11].

Com este mecanismo de garantia de atraso, é dada prioridade para pacotes que cheguem "na hora", em detrimento daqueles que chegaram cedo (antes da hora). Se, em um determinado fluxo, os pacotes chegam sempre antes do momento condizente com a taxa *r(f)*, este fluxo é efetivamente mais explosivo do que um fluxo que se comporte segundo a taxa negociada. A tendência para um fluxo mais explosivo do que o negociado, é que esta explosão seja suavizada. Tráfegos explosivos demandam mais memória de armazenamento nas filas de entrada dos servidores.

Assumido que cada fluxo tenha o seu próprio *buffer*, uma determinada fonte que não respeite a taxa negociada *r(f)*, será suavizada pelo algoritmo de relógio virtual. Assim, o servidor se comporta como um policiador de tráfego, pois explosões muito fortes (alta taxa) serão descartadas por estouro de *buffer*, e torna-se desnecessária a existência de uma entidade especializada em policiamento, verificando se as fontes são bem comportadas. é conveniente para a fonte fazer o seu próprio policiamento, de modo a não ter o seu tráfego prejudicado, seja pelo atraso introduzido, seja por perdas devido ao estouro da capacidade do *buffer*.

Esta garantia de atraso também gera uma propriedade muito importante: uma fonte mal comportada não prejudica nenhum dos demais fluxos que se utilizam do servidor. Esta propriedade é referenciada como propriedade de *firewall*[8].

Dado o momento *A(p)* (chegada) de um pacote, é garantido que o momento de sua saída ($L(p)$) será, no máximo, a soma de $A(p)$ mais o tempo de chegada de um pacote de tamanho *l(p)* na taxa *rf*, mais o tempo de transmissão do maior pacote permitido pelo canal de saída (capacidade C bps).

2.2 Pacotes de tamanho fixo

Em redes onde o tamanho do pacote é fixo, como em ATM, pode-se passar a utilizar número de pacotes como referência, ao invés de tamanho de pacotes em bits. Assim, os resultados anteriores podem ser reescritos, levando-se em conta a taxa do fluxo em pacotes por segundo ($\lambda(f)$) e a taxa do servidor, também em pacotes por segundo (γ).

A capacidade de transmissão através de transferência de blocos do ATM (*ATM block transfer - ABT*) foi padronizada pelo ITU-T [12]. Este padrão se baseou nas seguintes observações [2]:

- unidades de dados de protocolos de alto nível são enviados para transmissão e segmentados em diversas células, formando um bloco de células (explosão);
- uma pequena taxa de perda de células pode causar uma alta taxa de perda, para o protocolo de alto nível;
- a vazão média do protocolo de alto nível pode ser bem menor que a vazão apresentada por um bloco de células.

Assim, o conceito de bloco foi introduzido para representar um conjunto de células, que pode conter uma (ou mais) unidades de dados de um protocolo de alto nível. Um bloco é iniciado por uma célula que reserva a banda necessária para o bloco e é terminado por uma célula que libera esta banda passante. Assim, se não houver banda passante suficiente, todo o bloco é descartado. Isto assegura que não há ineficiência na utilização da rede, causado por transmissão de células de um bloco que teve uma célula descartada pois, ao ter uma única célula descartada, o protocolo de alto nível poderá solicitar a retransmissão de todo o bloco, não reaproveitando as células que chegaram corretamente. Também é utilizado o conceito de taxa de perda de bloco, ao invés de taxa de perda de célula.

3 Um modelo de tráfego apropriado

Um sistema de transmissão de vídeo representa as imagens como um modelo de *array* de pontos (*pixels*) onde cada ponto tem suas características armazenadas em 24 bits, em formato digital. A utilização de vídeo neste formato (sem compressão) não é razoável. O método mais usado para codificação de vídeo é o MPEG. Neste formato, existem três tipos de codificação de quadros de vídeo (I, P e B), cada qual com sua característica de tamanho, em número de bits.

Considerando um vídeo com um resolução de 640 x 480 pontos, e uma taxa de trinta quadros por segundo, sendo codificado em MPEG, obtem-se unidades de dados com tamanhos que variam de 10.000 bits a 300.00 bits [5], conforme mostra a Figura 1. Quando estas unidades são enviadas para a rede, geram um tráfego que varia de 0,3 Mbps até 9 Mbps. Numa rede ATM, cada unidade desta será segmentada em diversos pacotes de tamanho fixo.

Quando se utiliza um modelo de fluxo baseado em seqüência de explosões, pode-se garantir limites para a variação do atraso entre pacotes e entre explosões (blocos) e ter uma noção mais real da banda utilizada, já que pode existir uma grande diferença entre a taxa média gerada e a taxa durante a duração de uma explosão.

O fluxo pode, então, ser especificado:

- um fluxo é uma seqüência de explosões, cada qual formada por uma seqüência de pacotes. O primeiro e o último pacote de cada explosão são marcados, sendo que o primeiro pacote de uma dada explosão

i deve carregar a informação da taxa (λ_i pacotes por segundo) e, dependendo da implementação, seu tamanho (n_i).

- considerando-se a mesma explosão i, a diferença do tempo de chegada entre um pacote j qualquer ($A(j)$) e o primeiro pacote ($A(1)$) pode estar entre 0 (zero) e o valor dado por $(j-1)/\lambda_i$. Este valor representa o tempo necessário para se transmitir j - 1 pacotes, a uma taxa λ_i. Este é o limite para a variação do atraso entre pacotes.

- As explosões satisfazem uma separação mínima, entre si. Considerando-se duas explosões consecutivas, i e $i+1$, a diferença de tempo entre a chegada do primeiro pacote de cada uma delas deve ser, no mínimo, igual a n_i/λ_i. Este valor representa o tempo necessário para se transmitir n_i pacotes a uma taxa λ_i.

é importante notar que a taxa λ_i representa a taxa média dos pacotes da explosão e que redes especializadas em escalonamento de explosões são projetadas para preservar os limites de atraso especificados.

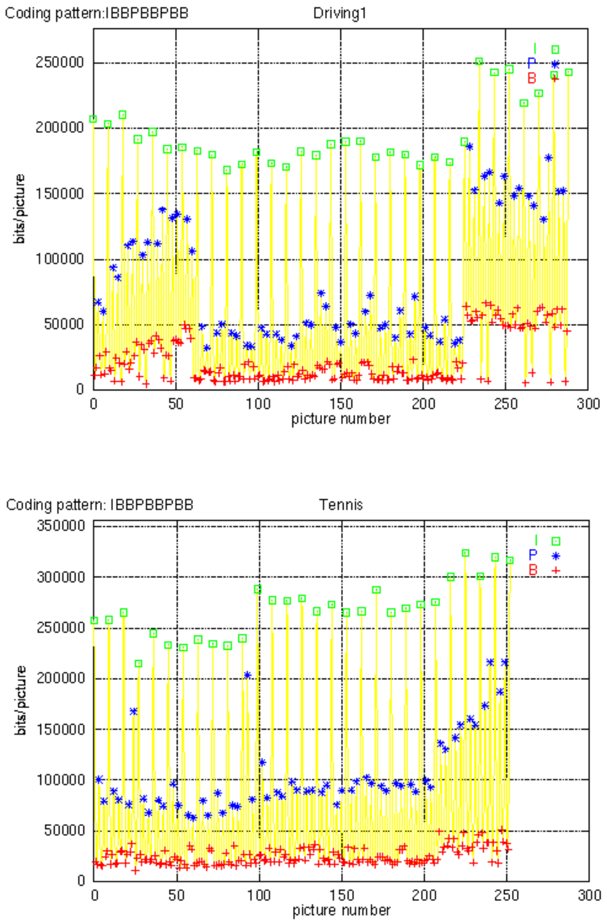

Figura 1: Tamanho dos pacotes de duas seqüências de vídeo.

4 Fluxos garantidos

Seja um servidor baseado em relógio virtual que tenha entrando fluxos de dados que satisfaçam as especificações dadas, ou seja, que cada fluxo seja uma seqüência de explosões que não se sobreponham no tempo; que cada primeiro pacote carregue a informação da taxa λ da explosão. Então [6]:

- O valor do relógio virtual do j-ésimo pacote em uma explosão i, $1 < j \leq n_i$, e' dado por:

$$P[i,j] = P[i,1] + \frac{j-1}{\lambda_i} \qquad (1)$$

- Se a capacidade do servidor não foi excedida desde que ele está ocupado (sem esvaziar suas filas), então o seguinte limite de tempo é aplicável para o j-ésimo pacote da explosão i, servida durante o período ocupado, para $i \geq 1, j = 1, 2, ..., n_i$:

$$L[i,j] \leq A[i,1] + \frac{j}{\lambda_i} + \frac{1}{\gamma} \qquad (2)$$

5 alteração das explosões

Seja uma rede com K + 2 nós. Destes nós, o nó 0 (zero) é a fonte de, o nó K + 2 é o destino e os nós 1 a K são comutadores, onde cada canal de saída é um servidor de relógio virtual. Um fluxo com garantias, que atravesse esta rede, poderá ter seu *jitter* reestruturado, e/ou a separação entre explosões retemporizada. Isto pode ocorrer pois, para cada nó, a seqüência de pacotes pode não satisfazer às restrições de *jitter* e tempo de separação. Como as restrições de tempo são assumidas por cada comutador, ao longo do caminho, os pacotes são atrasados pelos reguladores de fluxo para assegurar que as restrições de tempos sejam satisfeitas quando os pacotes estiverem prontos para serem servidos no próximo servidor.

Os reguladores de fluxo nos comutadores devem ser projetados para garantir (para $i \geq 1, j = 1, 2, ..., n_i$):

$$L[i,j] - P[i,1] \leq \frac{j-1}{\lambda_i} + \frac{1}{\gamma} \qquad (3)$$

Para a hora de saída dos pacotes que devem satisfazer as restrições de *jitter*, tem-se (para $i \geq 1, j = 1, 2, ..., n_i$):

$$0 \leq L[i,j] - L[i,1] \leq \frac{j-1}{\lambda_i} \qquad (4)$$

Através de (3) e (4), verifica-se que os pacotes de uma explosão i que estão perto de deixar um servidor de relógio virtual necessitam ser reestruturados, segundo mostrado em 5, antes que eles possam estar prontos para serem servidos no próximo servidor de relógio virtual, para $i \geq 1, j = 1, 2, ..., n_i$:

$$L[i,j] := max\{A[i,j], P[i,1] + \frac{1}{\gamma}\} \qquad (5)$$

Além da reestruturação para preservar o *jitter*, é necessário manter a temporização entre explosões. Assim, uma explosão i deve ser retemporizada quando ela torna-se pronta para ser servida, no próximo servidor, para $i \geq 1, j = 1, 2, ..., n_i$:

$$A[i,j] := max\{A[i,j], A[i-1,1] + \frac{n_{i-1}}{\lambda_{i-1}}\} \qquad (6)$$

No algoritmo apresentado em [6] a reestruturação (5) e a retemporização (6) são somente executadas para o primeiro pacote de cada explosão, sendo executada quando o pacote for o primeiro da sua fila, no próximo comutador. O atraso inserido por estas duas operações é insignificante.

Embora o regulador possa inserir atraso aparentemente sem motivo quando o servidor está vazio, é importante notar que essa reestruração e retemporização não afetam o atraso fim-a-fim no pior caso, ou seja, o limite superior para o atraso fim-a-fim continua o mesmo para explosões com ou sem alteração. O limite inferior será diferente.

6 Arquitetura do canal

Um comutador deverá ter uma arquitetura completa de canal para cada canal de saída. A arquitetura para um canal é mostrada na Figura 2. Para cada canal são mantidas filas individuais para cada fluxo, ou seja, cada fluxo com garantia tem alocado seu *buffer*. Para cada fluxo existe também um regulador.

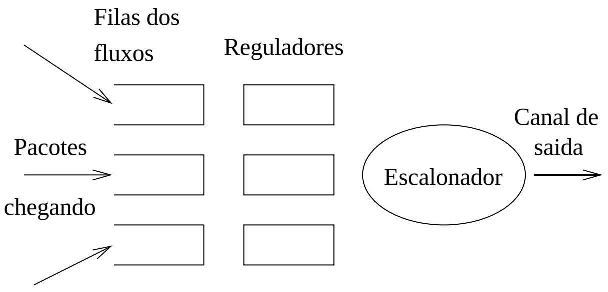

Figura 2: Arquitetura de canal

Finalmente, existe um escalonador para cada canal.

Um regulador tem por função básica guardar as informações guardadas no primeiro pacote de uma explosão e assegurar que o *jitter* e os limites de tempo de separação seja satisfeitos.

7 Tráfego ABR

Em redes projetadas para integração de serviços, há a necessidade de atender também a tráfegos que não necessitam de garantias de atraso para explosões. Este tipo de tráfego é referenciado como tráfego de taxa disponível (ABR - *available bit rate*). Para este tipo de tráfego, nenhuma taxa mínima é alocada. Assim, pode-se usar somente uma fila para acomodar todos os pacotes deste tipo de tráfego.

Se for necessário, pode-se alocar uma parte da banda passante para o tráfego ABT. Neste caso, pode-se alocar uma banda $\alpha\gamma$, com $\alpha < 1$, para o fluxo com garantias. Assim, teriamos o correspondente a $(1-\alpha)$ da banda do canal reservada para tráfego ABR.

8 Limites do atraso em explosões

Considerando que cada canal despacha seus pacotes sem erros e em ordem, e que a soma das taxas reservadas dos fluxos ativos não ultrapassa a capacidade do canal, em qualquer instante, pode-se chegar a limites para o atraso fim-a-fim. Seja a explosão i, $i \geq 1$, e o nóK. Os limites $D[i,1]$ inferiores e superiores para o atraso do primeiro pacote são mostrados a seguir:

$$D[i,1] \leq \frac{1}{\lambda_i} + (K-1)\max_{1\leq m\leq i}\{\frac{1}{\lambda_m}\} + \sum_{k=1}^{K}\beta_k \tag{7}$$

$$D[i,1] \geq (K-1)\frac{1}{\lambda_i} + \sum_{k=1}^{K}\beta_k \tag{8}$$

onde, $\beta_k = (1/\gamma_k) + \tau_{k,k+1}$ (segundos); γ_ké a capacidade do canal, do nók ao $k+1$ em pacotes/segundo; $\tau_{k,k+1}$é o tempo de propagação do nók ao nó$k+1$; $D[i,j]$é o atraso fim-a-fim do j-ésimo pacote da i-ésima explosão, medido do momento da chegada no nó 1 até a saída do nó$K+1$.

Para o atraso dos pacotes de uma mesma explosão, tem-se:

$$D[i,j] \leq D[i,1] + \frac{n_i}{\lambda_i} \tag{9}$$

Finalmente, o atraso fim-a-fim (D_i) de uma explosão i, medida do momento em chega o primeiro pacote no nó 1, até o momento em que o último pacote deixa o nó$K+1$, é limitado por:

$$D_i \leq D[i,1] + \frac{n_i}{\lambda_i} \leq D[i,1] + \theta_i \tag{10}$$

onde, θ_ié a duração máxima de uma explosão i, em segundos.

8.1 Parâmetro de QoS

Os parâmetros de QoS relevantes para serem manipulados por este serviço são:

- taxa máxima (λ_{max}) a ser reservada para uma explosão i ($\lambda_i \leq \lambda_{max}$, para todo m.
- atraso máximo fim-a-fim (D_{max}) para qualquer explosão do fluxo, a ser garantido pela rede.

No aspecto comercial, estes parâmetros podem ser usados para balizar a cobrança pelos serviços: quanto maior o valor da razão λ_{max}/D_{max}, maior o preço do serviço.

Para verificar a taxa máxima de uma explosão i, é suficiente que:

$$\frac{n_i}{\theta_i} \leq \lambda_{max} \tag{11}$$

Existe um compromisso entre o atraso fim-a-fim e a taxa alocada para uma determinada explosão. Se a taxa for muito baixa, não será possível garantir o atraso máximo negociado. Através de (7) e (10), chega-se a taxa de vazão mínima que deve ser alocada para uma explosão i, de modo a garantir um atraso máximo D_{max}:

$$\lambda_{i,min} = \frac{n_i + K}{D_{max} - \sum_{k=1}^{K} \beta k} \tag{12}$$

Assim, a taxa efetivamente reservada para para uma explosão ié dada por $max\{n_i/\theta_i, \lambda_{i,min}\}$. Uma renegociação pode vir a ser necessária, caso o valor de $\lambda_{i,min}$ tenda a ser maior do que o valor λ_{max} negociado, devido ao conflito criado.

9 Limitações

Este serviço garante ao gerador de fluxo o atraso e as propriedades de *firewall* mencionadas somente se as seguintes propriedades da rede e do fluxo entrante forem:

- o fluxo de entrada está de acordo com a especificação de fluxo descrita;
- os canais despacham os pacotes em ordem e sem perdas;
- os comutadores são confiáveis;
- a cada fluxo com garantias é alocado o seu próprio *buffer*;
- a capacidade do canal de saída nunca é excedida pela soma das taxas alocadas, em qualquer instante.

10 Garantias entre a rede e a aplicação

Uma vez garantida a qualidade de serviço na camada de rede, é necessário também prover estas garantias no caminho entre a camada de rede e a aplicação. Caso isto não seja feito, os atrasos e suas variações poderão comprometer os requisitos e limites da aplicação, pondo a perder o trabalho realizado no ambiente de rede.

Nos sistemas operacionais monousuário e monotarefa, cuja principal função é tão somente promover rotinas para acesso a disco (DOS) e demais periféricos de microcomputadores pessoais (IBM-PC), não existia proteção à memória e periféricos contra acessos indevidos. Enquanto se utilizava IBM-PC com DOS, podia-se ter acesso direto a toda memória disponível, e a periféricos, tais como placa de interface de rede e controladora de vídeo. Para estes sistemas, há alguns anos foram apresentadas arquiteturas de alto desempenho [13, 14], que sanavam a baixa eficiência na manipulação da memória, a falta de base para implementação de contadores de tempo, e a falta de mecanismos de acesso a rede, de bom desempenho.

Já os sistemas operacionais multiusuário e multitarefa, como o UNIX, se preocupam com a segurança dos dados na memória e controlam os acessos aos dispositivos de *I/O* da máquina, de modo a impedir que um processo mal comportado ou defeituoso venha a causar danos aos dados de outros processos/usuários. Assim, existe um núcleo (*kernel*) que gerencia o acesso a memória e a todos os periféricos da máquina. Qualquer acesso a um dispositivo de *I/O* como, por exemplo, a interface de rede, deve ser pedido ao *kernel*. A Figura 3 mostra a arquitetura de *I/O* de rede, típica do sistema operacional Unix.

Para utilização com aplicações multimídia, é necessário uma modificação/extensão da arquitetura de entrada/saída de rede. Esta modificação permite que aplicações multimídia com requisitos de QoS (atraso e vazão) possam ter suas condições mínimas satisfeitas [15].

A interface de rede no sistema Unix SunOS 5.3 é acessada pelo subsistema de rede implementado com *streams*. O caminho de escrita entre um processo do usuário e a interface de rede é feito da seguinte forma: Para a escrita, o processo do usuário faz uma chamada de sistema ***write()***, que copia os dados do usuário para um *stream buffer*; os dados são segmentados (se maior do que o MTU[1] da rede) e enfileirados para processamento pelo acionador de dispositivo; o acionador copia os dados para uma outra área de memória previamente mapeada pelo dispositivo controlador de DMA. Os dados são transferidos por DMA para a interface de rede.

Na recepção, uma otimização é feita, se houver um número de *buffers* de DMA livres acima de um limiar: a cópia do *buffer* de

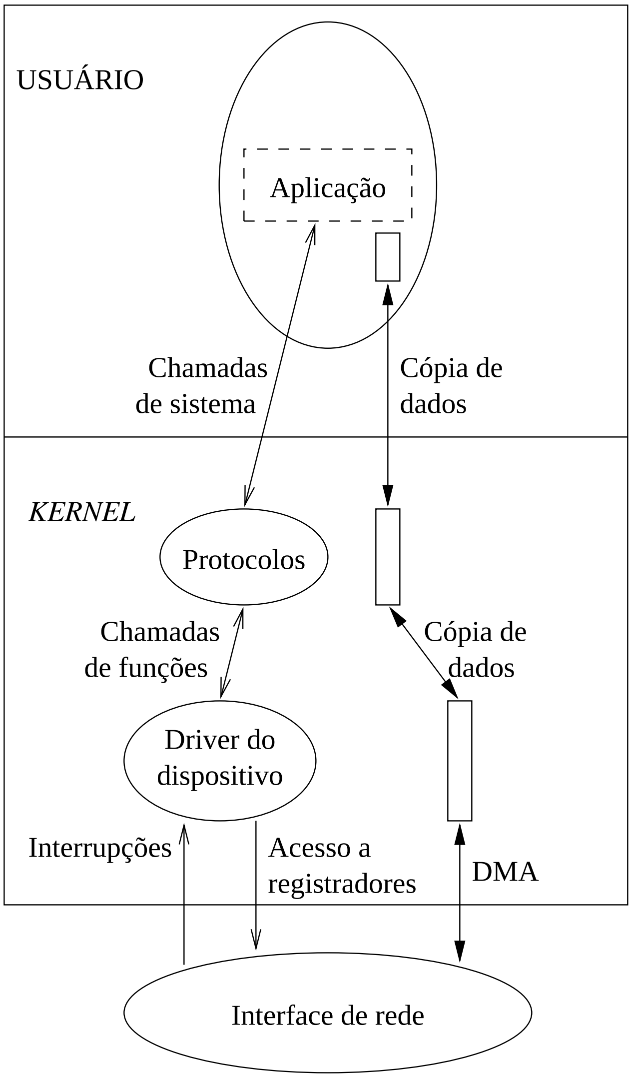

Figura 3: Arquitetura de *I/O* de rede no Unix

DMA para o *buffer* do subsistema de rede não é realizada. Mesmo assim, na transmissão permanecem as duas cópias de dados, realizadas via processador (movimentação memória a memória).

A cópia gera *overhead* de processamento inútil e demorado (acesso a memória é lento).

As chamadas de sistema são inadequadas para as aplicações que necessitam garantia de atraso. Existem dois tipos de chamadas: sícronas e assíncronas. Na chamada síncrona o processo fica "preso" até que o sistema

[1]MTU - *Max. Transfer Unit*: representa o tamanho máximo do pacote, em octetos, que pode ser transmitido pela rede.

atenda o pedido. Na chamada assíncrona, o sistema libera imediatamente o processo chamador, mas não garante o atendimento, retornando um aviso (erro) quando o pedido não pode ser atendido. Neste último caso, pode ser enviado um sinal ao processo do usuário, avisando quando a facilidade que foi negada anteriormente, estiver disponível. Pode-se notar que em ambos os casos a operação só é atendida quando o dispositivo acessado estiver desocupado. Desta maneira, a prioridade de tráfego torna-se virtualmente igual para todos os processos que estejam disputando o acesso ao dispositivo de *I/O*.

A necessidade de aplicações multimídia negociarem com a rede conexões com taxa reservada, já foi abordada por outros autores [16]. Para conseguir satisfazer aos parâmetros de QoS negociados, os pacotes pertencentes a uma conexão com taxa reservada tem que satisfazer a certas especificações de fluxo. Como o sistema operacional está no caminho entre a aplicação e a rede, ele tem que suportar controle de fluxo baseado em vazão (taxa).

Com a reestruturação da arquitetura, apresentada na Figura 4, várias modificações e adições foram feitas:

- melhoria na utilização dos *buffers*, de modo a reduzir a necessidade de cópia de dados na memória;

- foi criada uma "chamada rápida de sistema", que permite a aplicações obterem prioridade no atendimento de um envio de dados para a rede.

- o controle de fluxo baseado em vazão [17] pode ser feito com a inclusão de *threads* no núcleo do SO. Com isto, também fica reduzido o uso das chamadas de sistema.

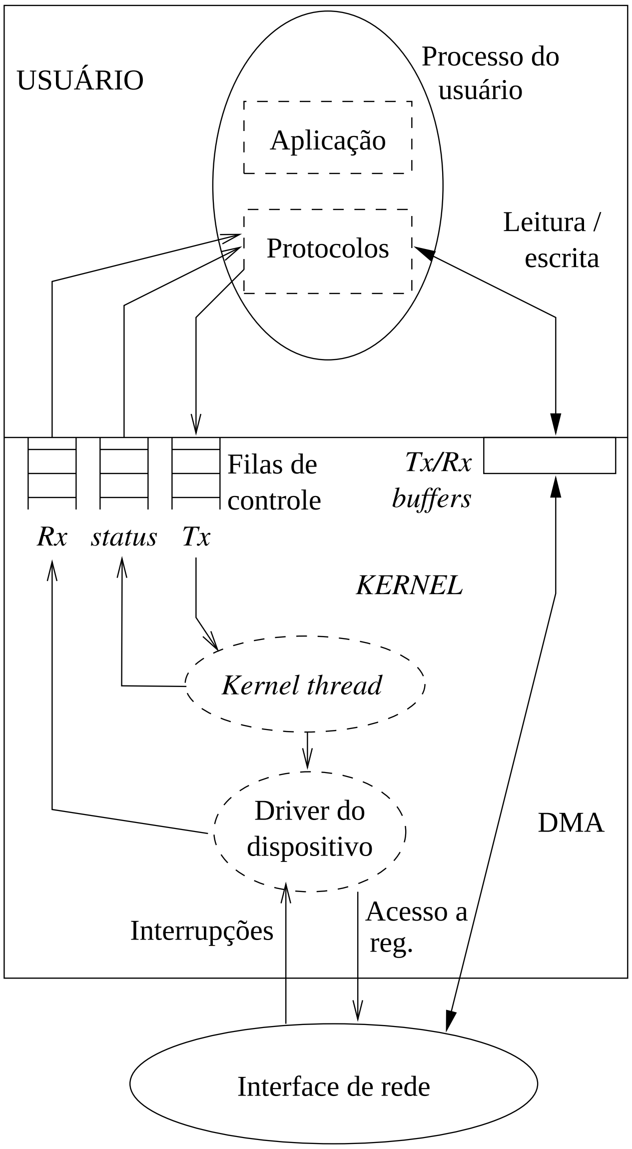

Figura 4: Arquitetura de *I/O* modificada

- acesso direto entre dispositivos de mídia e o subsistema de rede. Com esta inclusão, é possível enviar dados capturados por um transdutor diretamente para a rede, sem passar por nenhum processamento.

O estudo foi realizado a partir de uma aplicação de vídeo-conferência usada em mbone. Esta aplicação, chamada NV (*Network Video*), gera pacotes de vídeo seguindo o perfil mostrado na Figura 5. São usados pacotes RTP de 1040 octetos, cada qual sendo enviado através de uma chamada de sistema **send()**.

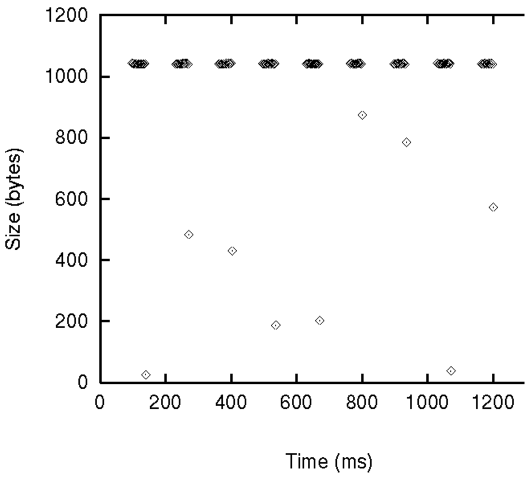

Figura 5: Padrão de envio de quadros de vídeo

10.1 *I/O* mais eficiente

As áreas de memória destinadas a armazenamento temporário para entrada e saída (*buffers*) foram classificadas segundo o mapeamento do endereçamento feito.

- usuário: memória alocada (endereço mapeado) para uso dos processos do usuário;
- núcleo: o espaço de memória é co-mapeado para o usuário e para o núcleo do SO. Esta área é útil para uma comunicação eficiente entre o processo do usuário e o núcleo.
- rede: esta área é co-mapeada para o usuário, núcleo e controlador de DMA. Assim, na passagem de dados do usuário para o núcleo e para a interface (via DMA), não é necessário fazer nenhuma recópia de memória.

A alocação destas áreas de memória é iniciada pelo usuário. Ao núcleo cabe fazer a alocação e os mapeamentos necessários. A gerência desta área fica por conta do usuário, que deve faze-lo de maneira eficiente, reutilizando-a da melhor forma possível.

O modo como é feita a alocação de *buffers* traz algumas limitações: uma vez que o código para acesso a rede deve ser centralizado e os *buffers* ficam centralizados, todas as operações de rede ficam dependentes deste código a nível de usuário. Assim, é necessário criar uma instância de protocolo para cada aplicação que abrir uma conexão na rede. Isto pode ser razoável, uma vez que não é comum abrir muitas aplicações de teleconferência, por exemplo.

A segurança do sistema não é diminuída com os acessos feitos diretamente pelo usuário, uma vez que estas alocações são feitas individualmente para cada conexão estabelecida, ou seja, as aplicações que utilizam acessos a rede ficam completamente isoladas. Para recepção, o hardware da interface de rede deve suportar demultiplexação a nível de enlace, de modo a poder reconhecer a qual processo se destina a conexão fim, e decidir para qual *buffer* deve ser enviado o dado.

Com estas modificações, já é possível perceber melhorias no tempo de latência[2]. A Figura 6 mostra o tempo de latência com o sistema normal (sem modificações) e o sistema com as extensões propostas, para o caminho de leitura, ou seja, na chegada de pacotes da rede para a aplicação.

[2]tempo para um pacote passar pelo SO. No caso, tempo entre a chegada do pacote na interface de rede e a entrega a aplicação.

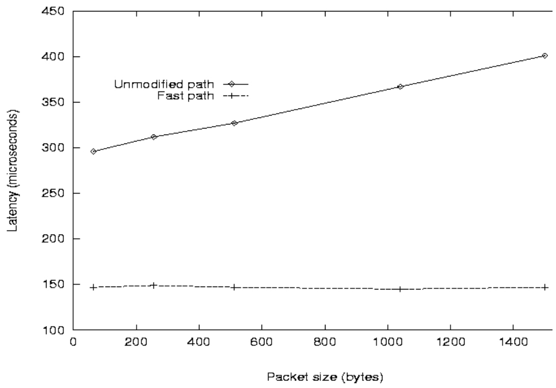

Figura 6: Latência por pacote no caminho de leitura.

10.2 Chamada rápida de sistema e *kernel thread*

A chamada rápida de sistema é, na verdade, um complemento ao *kernel thread*. Uma vez que uma determinada conexão é feita baseada em uma determinada banda passante negociada, cabe ao *thread* cumprir com a taxa especificada. O controle desta taxa é feita baseada em relógio virtual. A cada vez que roda, a rotina do *thread* envia para a fila da interface de rede um número de pacotes necessários para manter a taxa dentro do limite especificado. Mas, como o *thread* inicia seu algoritmo em períodos de tempo não sincronizados com a aplicação, pode ocorrer a necessidade de enviar um pacote (ou mais) antes do ciclo de *thread* ocorrer. Assim surge a chamada rápida de sistema.

Esta nova chamada permite ao usuário ativar o envio forçado do primeiro pacote de sua fila, esvaziando-a mais rapidamente. Não há uma troca de prioridade dentro da fila daquela aplicação. Toda a fila fica com maior prioridade, e o primeiro pacote desta fila é enviado imediatamente para a fila da interface de rede.

Cabe a aplicação reconhecer se é necessário ou não acionar uma chamada rápida de sistema. Para isto existe um indicador (*flag*) que é manipulado pelo núcleo e que pode ser lido pela aplicação. Este indicador é ativado, ao fim de um ciclo de *thread*, caso o relógio virtual da fila (ou fluxo) esteja menor do que o valor em que se iniciará o próximo ciclo de *thread*. Como o valor do relógio virtual representa o momento em que o pacote deve ser transmitido para manter a taxa negociada, o fato deste valor estar menor que o momento em que se iniciará o próximo *thread* significa que o atendimento aos pacotes está atrasado, e que não se pode esperar pelo próximo ciclo de *thread*, sob pena de não atender a taxa negociada.

Um *kernel thread* pode ser dedicado ou não dedicado[3]: no dedicado, existe um *thread* para cada conexão, ou seja, é dedicado ao processo que é seu cliente. Nesta modalidade, o thread é criado no mesmo contexto do processo do usuário, o que traz algumas vantagens: pode acessar toda a memória do cliente; é possível fazer *upcall*[4]; permite uma grande cooperação com o usuário.

No *thread* não dedicado não é assumido o contexto do espaço de endereçamento do usuário, podendo ter mais de um cliente. A eficiência deste modelo é maior, por produzir menor troca de contexto, quando há vários clientes.

A Figura 7 mostra a variação da latência com o tamanho do pacote enviado, para o sistema normal, utilizando somente *thread* dedicado e utilizando chamada rápida. A chamada rápida possui, como era de se esperar, a menor latência. Nesta medida, o pacote enviado era o único na fila de chamadas, sendo atendido imediatamente, no caso da chamada rápida.

Na Figura 8, é mostrada a variação da

[3]este modelo é chamado de *lightweight* pelos seus inventores.

[4]o *thread* usa rotinas da aplicação do usuário.

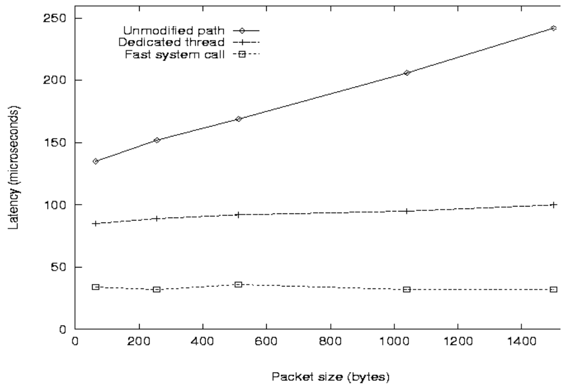

Figura 7: Latência na escrita, variando-se o tamanho do pacote.

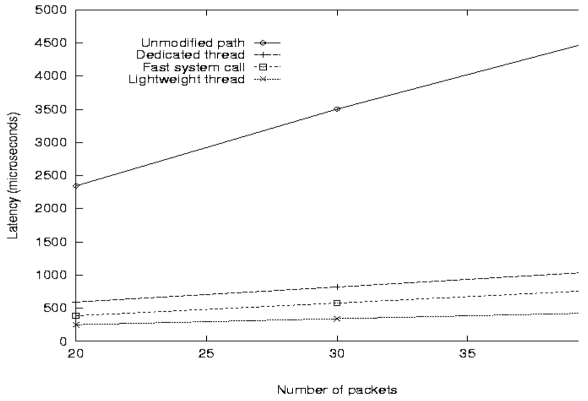

Figura 8: Latência na escrita, variando-se o número de pacotes por explosão.

latência com o tamanho de explosões de pacotes (20, 30, 40). Os pacotes são, cada um, de 1040 octetos, oriundos da aplicação de vídeo, NV, já citada. Nesta figura (8), a menor latência ficou com o *thread* não dedicado. Isto se deve ao amortecimento da única troca de contexto necessária para cada explosão de pacotes deste método. Assim, este é o método mais eficiente para manipular grandes quantidade de envios.

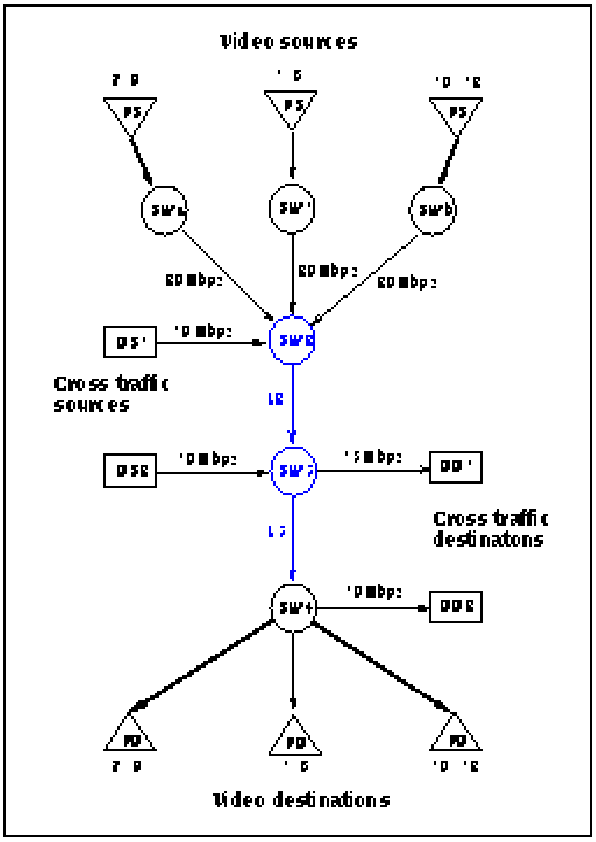

Figura 9: Topologia da rede de simulação.

11 Resultados experimentais

Para testar a garantia de QoS, inicialmente foram feitos testes somente no nível de rede [6]. Para isto foi utilizada a topologia de rede mostrada na Figura 9. Nesta topologia, os canais L2 e L3 tem a mesma capacidade C. No experimento, esta capacidade foi escolhida de modo que a soma das taxas de pico dos 12 fluxos de vídeos utilizados igualassem a 80% da capacidade C. Para os geradores de tráfego CS1 e CS2, foram utilizadas fontes de Poisson com taxa entre 20% a 21% da capacidade C.

Está listada na Tabela 1 a relação dos vídeos utilizados para estas medidas, com respectivas características de vazão (λ) mínima, máxima e média.

Tabela 1: Taxas dos vídeos utilizados nas medidas

Vídeo	λ (Mbps)		
	min	max	med
Terminator	0,14	3,86	1,15
ParentsSon	0,37	5,97	1,51
RedsNightmare	0,089	3,62	0,75
Student	0,48	2,47	1,27
Driving	0,17	8,48	1,88
Airwolf 2	0,14	3,31	0,89
Simpsons I	0,14	2,60	0,92
Canyon	0,076	0,70	0,28
FlowerGarden	1,39	13,25	5,04
UnderSiege	0,17	2.02	0,59
StarTrek II	0,28	1,15	0,62
Energizer	0,17	2,34	0,76

Todas as fontes de dados são acionadas simultaneamente para realizar as medidas. A Figura 10 mostra o atraso fim-a-fim para o primeiro pacote de cada explosão, medido para o vídeo *Energizer*. O atraso fim-a-fim, medido para cada quadro do mesmo vídeo, está ilustrado na Figura 11.

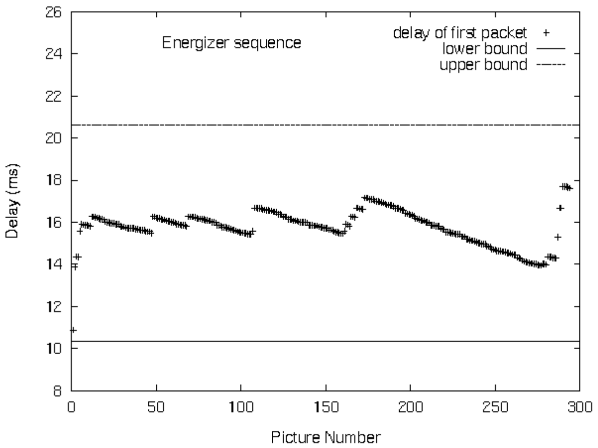

Figura 10: Atraso fim-a-fim do primeiro pacote de cada explosão do vídeo *Energizer*.

A seguir serão mostrados alguns resultados de medidas feitas para testar as modificações

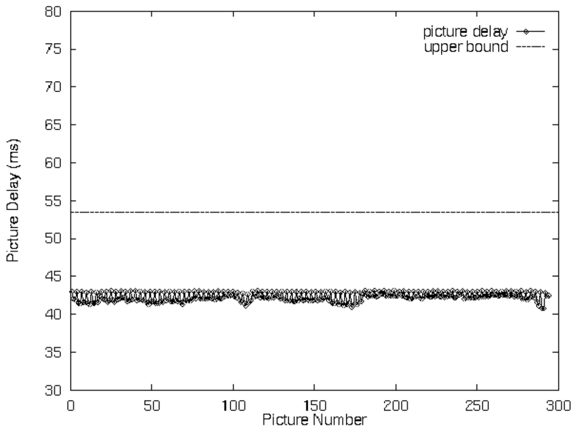

Figura 11: Atraso fim-a-fim para os quadros do vídeo *Energizer*.

introduzidas no sistema operacional [15].

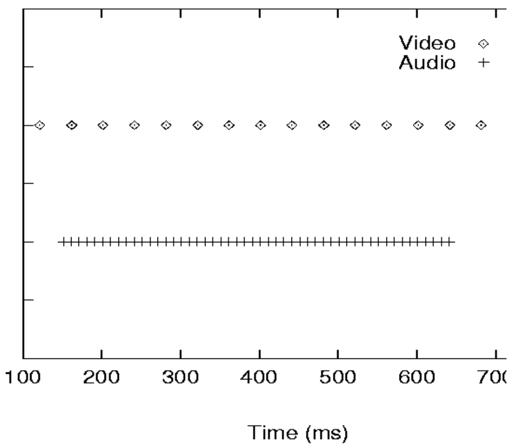

Figura 12: Amostra da geração dos pacotes de áudio e vídeo.

A aplicação utilizada para realizar estes testes contém vídeo, som e passagem de dados de um texto em *postscript*. O teste consiste em operar normalmente com áudio e vídeo e enviar o arquivo postscript de 60 Koctetos durante o uso da vídeo conferência, para um sistema não alterado. A seguir, o mesmo teste é realizado com a nova arquite-

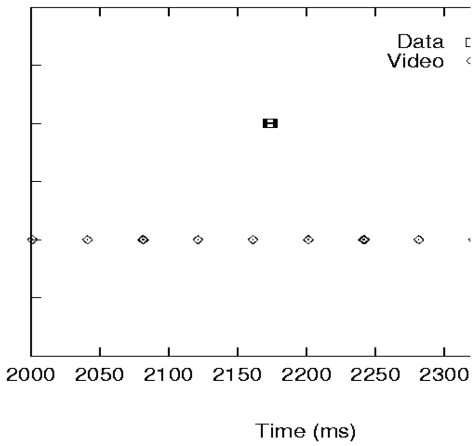

Figura 13: Envio do pacote de dados e geração de vídeo.

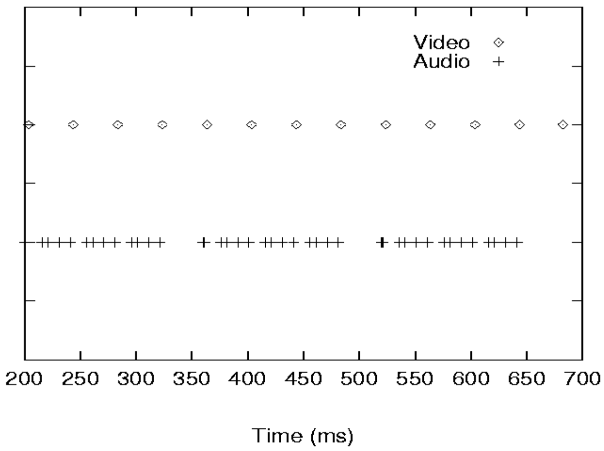

Figura 14: Chegada dos pacotes de áudio (suavidade perdida) e vídeo.

tura.

A aplicação de vídeo tem maior prioridade que a aplicação de áudio que, por sua vez, tem maior prioridade que a aplicação de envio de texto, em termos de escalonamento de CPU.

A Figura 12 mostra a seqüência original de pacotes de áudio e vídeo gerados pela aplicação. A Figura 13 mostra o momento em que o arquivo de texto é enviado, tendo, no mesmo gráfico, a geração de vídeo para aquele intervalo.

Utilizando uma rede com banda passante reservada de 5 Mbps para os testes, mediu-se os tempos de chegada dos pacotes no destino. A Figura 14 mostra a chegada dos pacotes de áudio e vídeo. Nota-se claramente que a suavidade no fluxo de áudio foi perdida. A Figura 15 mostra a distribuição, no tempo, da chegada dos pacotes de áudio, na estação destino.

O mesmo fenômeno de perda de suavidade ocorre com os pacotes de vídeo, quando o arquivo de dados é enviado (Figura 16). Por não haver prioridade na fila de rede, uma vez que o arquivo de texto é enviado, ele ocupa a fila de saída da interface de rede até que tenha sido todo despachado pela rede. Embora não esteja mostrado, o fluxo de áudio também é prejudicado.

Com a utilização das técnicas descritas neste trabalho, os resultados dos testes demonstram que é imprescindível adotar técnicas semelhantes, para utilizar sistemas multimídia em sistemas operados por sistemas operacionais como o Unix.

A Figura 17 mostra a nova distribuição da chegada dos pacotes de áudio, no tempo. Nota-se, comparando com a Figura 15, que a chegada dos pacotes ficou mais concentrada em torno de 10 ms.

A figura 18 mostra como ficou o tempo de chegada dos pacotes de vídeo, utilizando o sistema operacional estendido.

12 Conclusões

Partindo do princípio de que a garantia de qualidade de serviço visa a atender ao

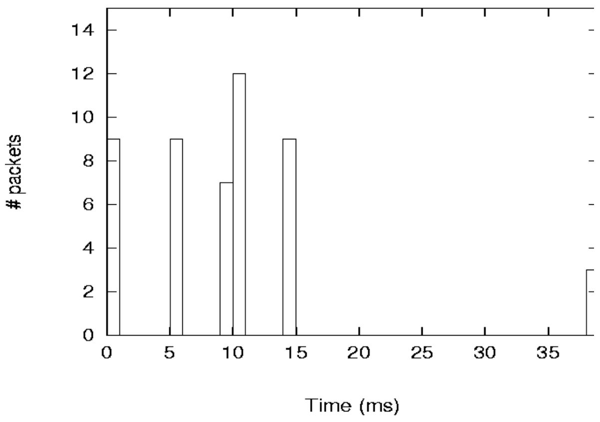

Figura 15: Distribuição no tempo da chegada dos pacotes de áudio.

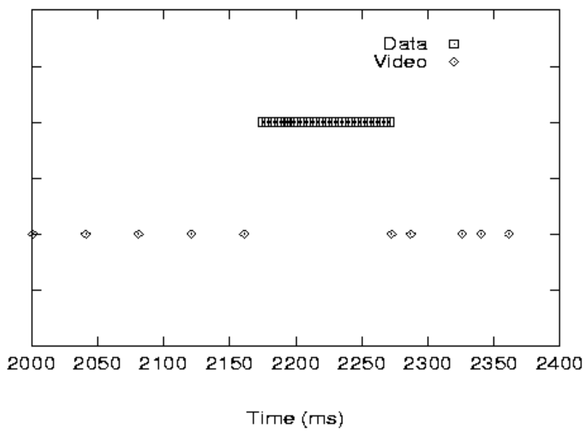

Figura 16: Quebra da suavidade do fluxo de vídeo, devido ao arquivo de texto.

usuário, e de características do fluxo de áudio e vídeo de uma aplicação multimídia típica, foi apresentada uma solução para garantir qualidade de serviço aplicação a aplicação, em termos de atraso máximo e variação de atraso.

Esta solução envolve o projeto de uma rede apropriada para escalonamento de explosões de pacotes, com escalonamento baseado em relógio virtual, e alterações no sistema operacional Unix (SunOS 5.3). O desenvolvimento relativo à rede consiste em especificar um novo modelo de fluxo que manipule com as características do fluxo de vídeo e o projeto de uma rede que reconheça este modelo de fluxo, garantindo limites de atraso de variação de atraso para os fluxos com garantias de QoS. As modificações no sistema operacional dotaram o subsistema de rede da capacidade de manipular com fluxos que tenham requisitos de QoS (banda reservada, atraso máximo e variação de atraso limitada).

As medidas mostradas revelaram que o objetivo foi alcançado, e que só é possível utilizar confortavelmente aplicações multimídia em máquinas de uso geral e compartilhado, se tais alterações/extensões forem realizadas.

Referências

[1] S. S. Lam, “Providing application-level QoS,” Tech. Rep. 96-26, Dpto of Computer Sciences, The University of Texas at Austin, Oct. 1996.

[2] S. S. Lam and G. G. Xie, “Real-time block transfer under a link sharing hierarchy,” in *Proc. IEEE INFOCOM'97*, (Kobe, Japan), Apr. 1997.

[3] D. K. Yau and S. S. Lam, “An architecture towards efficient OS support for distributed multimedia,” Tech. Rep. 95-36, Dpto of Computer Sciences, The University of Texas at Austin, July 1995.

[4] S. S. Lam and G. G. Xie, “Burst scheduling networks. technical report tr-94-20 (abbreviated version),” in *Proc. IEEE INFOCOM'95*, Apr. 1995.

[5] S. C. Simon S. Lam and D. K. Yau, “An algorithm for lossless smoothing of MPEG video,” in *Proc. ACM SIGCOMM'94*, (London, England), Aug. 1994.

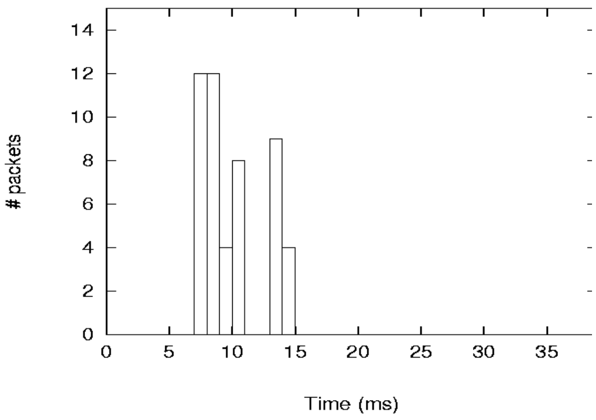

Figura 17: Nova distribuição dos pacotes de áudio, no tempo.

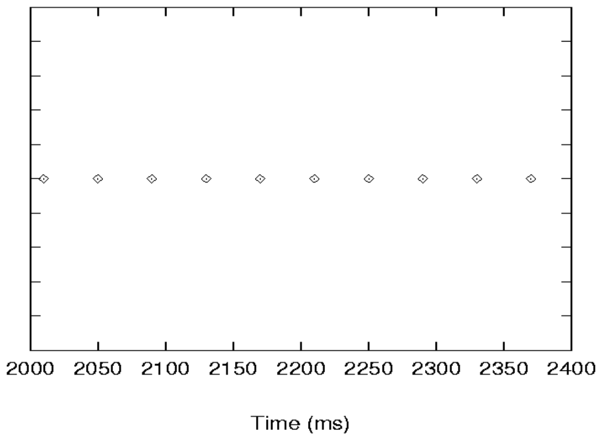

Figura 18: Chegada dos pacotes de vídeo, durante o envio do arquivo de texto, com o SO estendido.

[6] S. S. Lam and G. G. Xie, "Burst scheduling networks," Tech. Rep. 94-20 (3rd revision), Dpto of Computer Sciences, The University of Texas at Austin, Oct. 1996.

[7] L. Zhang, "Virtual clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. on Computer Systems*, vol. 9, pp. 101–124, May 1991.

[8] S. S. Lam and G. G. Xie, "Burst scheduling networks: Flow specification and performance guarantees," in *Proc. NOSSDAV'95*, (Durham, New Hampshire), pp. 303–306, Apr. 1995.

[9] S. S. Lam and G. G. Xie, "Group priority scheduling," in *Proc. IEEE INFOCOM'96*, Mar. 1996.

[10] S. S. Lam and G. G. Xie, "Group priority scheduling," Tech. Rep. TR-95-28 (3rd revision), Dpto of Computer Sciences, The University of Texas at Austin, Jan. 1997.

[11] G. G. Xie and S. S. Lam, "Delay guarantee of virtual clock server," *IEEE/ACM Trans. Networking*, vol. 1, pp. 683–689, Dec. 1995.

[12] Rec. I.371, *Traffic Control and Congestion Control in B-ISDN*. ITU–T, Genebra, 1992.

[13] L. F. Baginski and O. C. M. B. Duarte, "Um modelo de implementação de alto desempenho para sistemas abertos," in *IX Congresso da Sociedade Brasileira de Telecomunicações*, (São Paulo, SP), pp. 19.3.1–19.3.5, Sept. 1991.

[14] C. V. N. de Albuquerque, M. L. D. Lanza, and O. C. M. B. Duarte, "Sistemas de comunicação abertos em redes locais: Uma implementação e medidas de desempenho em ambientes de computadores pessoais," in *XIII Congresso Sociedade Brasileira de Computação (SEMISH)*, (Florianópolis), pp. 327–329, Sept. 1993.

[15] D. K. Yau and S. S. Lam, "Operating system techiniques for distributed multimedia," in *Proc. IS&T/SPIE MMCN*, Jan. 1996.

[16] C. Partridge, "A proposed flow specification," in *Internet Request for Comments - RFC-1363*, 1992.

[17] D. K. Yau and S. S. Lam, "Adaptative rate-controlled scheduling for multimedia applications," in *Proc. ACM-MM'96*, (Boston, Massachusetts, USA), Nov. 1996.